BarnOwl: Secure Comparisons using Silent Pseudorandom Correlation Generators

Sameer Wagh*

Devron Corporation

swagh@alumni.princeton.edu

June 21, 2022

Abstract

Recent advances in function secret sharing (FSS) have led to new possibilities in multi-party computation in the pre-processing model. Silent Pseudorandom Correlation Generators (Crypto '19, CCS '19, CCS '19, CCS '20) have demonstrated the ability to generate large quantities of pre-processing material such as oblivious transfers and Beaver triples through a non-interactive offline phase (with an initial set-up). However, there has been limited protocols for pre-processing material such as doubly authenticated bits (daBits, IndoCrypt'19) and extended doubly authenticated bits (edaBits, Crypto '20) which are critical for state-of-the-art secure comparison protocols over arithmetic secret sharing.

In this work, we propose new protocols in a 3-party computation model for these two cryptographic primitives – daBits and edaBits. We explore how advances in silent PCGs can be used to construct efficient protocols for daBits and edaBits. Our protocols are secure against a single corruption in both the semi-honest and malicious security models. Our contributions can be summarized as follows:

(1) New constant round protocols for generating daBits and edaBits. We achieve this by constructing an efficient 3-party oblivious transfer protocol (using just 2 rounds of computation) and using it to build efficient protocols for daBit and edaBit generation.
(2) We extend the above semi-honest protocol to achieve malicious security against an honest majority. We use a standard cut-and-choose approach for this. This improves the round complexity of prior edaBit protocols from $O(\log_2 \ell)$ to a constant, where ℓ is the bit-length of the inputs.
(3) Finally, to understand when the above protocols provide concrete efficiency, we implement and benchmark the performance of our protocols against state-of-the-art implementation of these primitives in MP-SDPZ. Our protocols improve the throughput of daBit generation by up to 10× in the LAN setting and 5× in the WAN setting. Comparing the performance of edaBit generation, our protocols achieve 4× higher throughput in the LAN setting and 32× higher throughput in the WAN setting.

It is known that silent PCGs are compute intense and thus the performance of these new protocols can further be improved using works such as CryptGPU (S&P '21), Piranha (USENIX '22) that significantly improve the local computation in MPC protocols.

*Work done, in part, while at UC Berkeley.

1. Introduction

Multi-party computation (MPC) is the study of cryptographic protocols which enable a set of non-colluding parties to compute a function of their inputs without revealing them. In two seminal works in the 1980s [1, 2], Andrew Yao first formalized the domain of secure computation for a specific function computation – secure comparison between two inputs. The problem came to be known as Yao's Millionaire problem and led to a growing interest in the field of secure computation. Another seminal work [3] demonstrated that two party computation can be generalized to multi-party with malicious adversaries (parties that can arbitrarily deviate from the protocol specification). Overall, the theoretical foundations [4–7] of securely computing arbitrary functions across various adversarial and computational models have since been well understood.

Initially, MPC protocols were studied primarily for their theoretical understanding. However, with reducing computational costs and algorithmic improvements, interest in the practical performance of MPC protocols has been growing. Starting with [8], there emerged a long line of work in improving the concrete efficiency of MPC protocols [9–13, 13–23]. An important approach in improving the overall performance of MPC protocols has been to design protocols in the pre-processing model. In the pre-processing model, computing a certain function on a specific input is split into two components (1) an offline component that performs a similar or related computation but on data that is independent of that specific input and (2) an online component that uses the result of such offline computation along with the specific inputs to compute the function efficiently. This approach has a number of benefits - (1) the offline computation can be parallelized, (2) it can be produced even before the actual data to be computed on is available, (3) it improves the overall performance of the computation, and/or (4) it can be offloaded to external agents. This idea dates back to the seminal work by Beaver [24] where two inputs can be multiplied securely by assuming access to a random pre-multiplied triple of values. Since then, a number of works have focused on improving the performance of MPC protocols using this model [13–18, 25].

Analogous to how Beaver triples are used for implementing secure multiplications in the pre-processing model, there are other primitives that have been useful for converting other secure computation building blocks into the pre-processing model. One notable example is *Oblivious Transfers (OT)* [26], a primitive known to be complete for MPC [27, 28]. Oblivious transfer enables one party (called Receiver) to receive only a single value among multiple values held by another party (called Sender) without revealing the index of the value. Advances in OTs have led to a primitive known as *OT Extensions* that allows a few base OTs (also know as seed OTs) to be expanded into a large number of OTs [29–31] cheaply (using symmetric-key primitives). More recently, a line of work has proposed *silent* OT extensions [14, 15] that can run the expansion phase locally, i.e., non-interactively among the parties.

The problem of generating pre-processing material in secure computation has been widely studied for building blocks such as multiplications, oblivious transfers. However, there have been few works that focus on primitives that are useful for secure comparisons. Part of the reason is that comparison has a complex structure and is not amenable to having simple correlated randomness as pre-processing. Recently, a few primitives have explored pre-processing tailored for secure comparisons. Rotaru and Wood [32] propose a primitive known as doubly authenticated bit a.k.a. daBit that securely generates a bit shared in two different forms of secret sharing – arithmetic and Boolean. This primitive helps in converting a secure comparison problem (where inputs are in arithmetic sharing) into a Boolean problem, where the protocol for comparison is significantly more efficient. Escudero *et. al.* [22] further improve upon this and propose a primitive known as extended doubly authenticated bit a.k.a. edaBit. This primitive securely generates a random value (as arithmetic secret shares) and its bit-decomposition (as Boolean shares) and thus, further improves the performance of secure comparison protocols. However, despite these recent advances, the overhead of these primitives is still a performance bottleneck in applications of secure computation.

Our contributions. In this work, we propose novel protocols for these two primitives – daBit and edaBit, which enable state-of-the-art protocols for secure comparison in a pre-processing model. We focus on the standard 3-party computation framework used in a number of prior works [25, 33–36] and focus on arithmetic circuits over rings, though the results extend to prime fields. Our main contributions are:

New protocols for daBits, edaBits. We propose new protocols for generating daBits and edaBits in a 3-party computation model. Our protocol for edaBits achieves a constant round complexity with high concrete efficiency.

Prior state-of-the-art works [22, 37] involve a round complexity which is logarithmic in the bit length of the data size. We make black-box use of silent OT correlation generators to achieve these improved results.

Semi-honest and malicious security. We construct protocols that are secure against a *single corruption in both the semi-honest and malicious setting*. We use standard cut-and-choose techniques for malicious adversaries and construct an efficient verification scheme for edaBits (used in the cut-and-choose).

Concrete efficiency. We implement our protocol and demonstrate that it improves the throughput of daBit generation by over 10× in the LAN setting and 5× in the WAN setting. When comparing the performance of edaBit generation, our protocols achieve 4× higher throughput in the LAN setting and 32× higher throughput in the WAN setting. Our gains largely stem from a simple protocol construction, reduced round complexity, and improvements in silent OT generation.

1.1. Applications of daBits, edaBits

daBits and edaBits are useful for generic secure computation and are used in many secure computation libraries and frameworks [25, 37–40]. They are also being used in other privacy-preserving applications, and we describe some concrete applications below.

Application for Secure Comparison. Secure comparison is an important primitive in secure multi-party computation. State-of-the-art frameworks for secure computation typically use a hybrid approach – using a combination of arithmetic, Boolean, and garbled circuits for secure computation. Such frameworks require efficient protocols for share conversion to convert between different representations. These conversion techniques directly rely on the daBits and edaBits primitives to convert between arithmetic and Boolean sharing. Examples of frameworks that use these primitives include ABY [40], ABY3 [35], MP-SPDZ [41], Mystique [42], Manticore [43]. Similarly, state-of-the-art comparison protocols in the arithmetic blackbox model [22, 23, 44] all use these primitives. Thus any improvement in the performance of generating daBits, edaBits directly improves the performance of these frameworks.

Application for Secure Federated Learning. Federated Learning (FL) is a machine learning technique that enables a single entity to train a neural network across multiple decentralized edge devices such that the private data at each edge device stays local to the device. This is achieved by aggregating model gradients at the centralized server. However, gradients are shown to reveal information on the training data and thus secure federated learning aims to hide these gradients using multi-party computation.

Given that the gradients are now secret shared, an important challenge is to ensure that the gradients are bounded within a certain range. The first technique is to receive data over smaller bit-lengths and then convert that into a larger domain for aggregation using share conversion in MPC. Any improvements in the generation of such correlated randomness directly improves the performance of state-of-the-art secure FL protocols such as Prio [45] and Prio+ [37]. The second technique to ensure that is the use of zero-knowledge range proofs and is used in a number of works [46]. There are also works that use edaBits to improve the performance of zero-knowledge proofs [42, 47].

Other Applications. While the primitives of daBits and edaBits are general and widely used in the above applications, they are also used in a number of other privacy-preserving applications such as decentralized cryptocurrency exchanges [48], privacy-preserving machine learning [44, 49], and privacy-preserving math libraries [50].

1.2. Technical Overview

We work in a 3-party setting with an honest majority of corruption. At the heart of our construction is an interplay between a 2-out-of-3 secret sharing and a 3-out-of-3 secret sharing. The goal is to generate shares in two different domains – arithmetic and Boolean. The shares, both arithmetic and Boolean, of 0 and 1 can be locally generated non-interactively and our main insight is that the 2-out-of-3 shared Boolean values can simply be used to "privately select" the appropriate arithmetic share in a 3-way oblivious transfer.

Thus, our first building block is a 3-party oblivious transfer functionality that generalizes the standard 2-party OT functionality. In our formulation, each party simultaneously acts as a sender as well as a receiver of an oblivious transfer (OT). The functionality is presented in Fig. 1. Each party P_i thus acts as an OT sender for the party P_{i+1} and as a receiver for party P_{i-1} where the indices are considered modulo 3.

$\mathcal{F}_{\mathsf{3OT}}$ and Malicious Security. We construct a protocol that securely emulates this 3-party OT functionality $\mathcal{F}_{\mathsf{3OT}}$. The parties use a random OT correlation (gen-

erated "silently") to have the sender mask it's inputs and the receiver select the appropriate string. However, this operation is performed cyclically, each party acts as the receiver for strings held by the next party. This structure ensures that the malicious party only receives its outputs from the next party and thus allows a maliciously corrupt party to introduce only two types of errors. First, a malicious party can arbitrarily corrupt the outputs of the "previous" party, and second, it can provide inconsistent sharing (i.e., non-conforming replicated secret sharing; refer to Section 3 for details on consistency of sharing). Given the honest majority of parties, we know that at least one party generates the correct output. Combined with the redundancy of replicated secret sharing, this allows us to construct a maliciously secure protocol. We then use Π_{3OT} to construct a protocol for daBits as described below.

Constructing daBits from Π_{3OT}. Note that it is possible to *non-interactively* generate correlated randomness in the above 3-party computation model, in both the semi-honest and malicious adversarial models. This is achieved by assuming access to pairwise shared keys (refer to Section 3 for details). We focus on two different types of correlated randomness, a 3-out-of-3 sharing and a 2-out-of-3 sharing. We then generate a 3-out-of-3 arithmetic sharing of 0 and 1 (call these $x = 0$ and $y = 1$ respectively) and a 2-out-of-3 Boolean sharing of a random bit b. The main insight here is to use $\mathcal{F}_{\mathsf{3OT}}$ and receive a 2-out-of-3 arithmetic sharing of x if $b = 0$ and y if $b = 1$. In other words, we generate a 2-out-of-3 arithmetic sharing of b. Combined with the original 2-out-of-3 Boolean sharing of b, we have successfully generated one daBit.

To extend this to be secure against a maliciously corrupt adversary, we use two techniques. The first is a hashing-based consistency check to ensure that the generated correlated randomness is consistent. The second technique is known as cut-and-choose [22,51,52] in the literature and ensures correctness of a large batch of generated correlated randomness. The central is to generate many daBits (possibly with errors) and perform the following steps:

(1) Shuffle the set of generated daBits.
(2) Open a few daBits and check for correctness.
(3) Add the remaining daBits into a number of bins. In each bin, use all the daBits to verify the first daBit without revealing.
(4) Finally, if all checks succeed, output the first daBit in each bin.

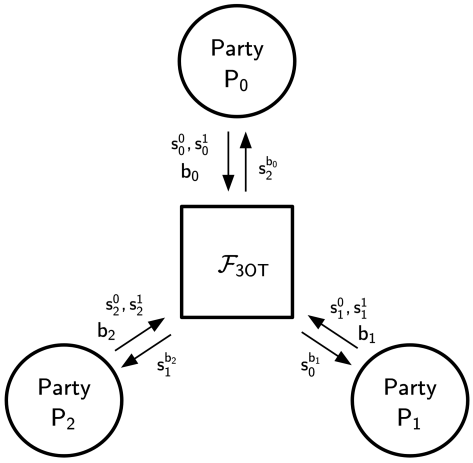

Figure 1: 3-Party Oblivious Transfer Functionality. Note that the output of the oblivious transfer is "clockwise", i.e., each party sends to the "next" party and receives from the "previous" party. Refer to Section 3 for details.

For the right parameters (such as those set in Steps 2 and 3 above), a combinatorial argument bounds the probability of generating any incorrect daBit to a statistically low probability. Step 3 involves a crucial verification routine Π_{Verify} which is described in Section 4.

Constructing edaBits from daBits. The final component is the generation of edaBits using daBits. We present two different approaches here. The first is to use a naïve approach where multiple daBits are simply locally combined to generate a single edaBit. This is conceptually simple and thus requires only generating a large number of daBits. This protocol is ideally suited for the semi-honest adversarial model.

In the malicious adversarial model, the former approach requires generating maliciously secure daBits and then locally combining them. The second approach provides improved concrete efficiency for the generation of edaBits. This approach generates semi-honest edaBits (possibly with errors) using the semi-honest daBit generation routine which has much higher throughput than malicious daBits. A cut-and-choose protocol is then run over these semi-honest edaBits. This amortizes the cost of the shuffle and reduces the size over which cut-and-choose routines are run by a factor of $32 - 64\times$. This final component requires a verification routine which can verify one edaBit using another without opening. This is Π_{Verify2} and is described in Section 6.1.

2. Preliminaries

This section summarizes the existing work on secure comparison protocols in the pre-processing model and PCG constructions we use in our protocols.

2.1. Secure Comparison Protocols & Primitives

The problem of secure comparison has been studied for almost as long as the field of secure computation itself. The primitive, that securely compares two or more inputs, is fundamental to almost all of modern day computing. First stated as the Yao's millionaires' problem in a seminal work by Andrew Yao [1,2], research on this problem has made tremendous progress [8–12, 21–23].

In the pre-processing model, the online protocol for computing secure comparisons has been nearly the same. More specifically, the online protocols all use either Boolean computation using a variant of a ripple carry adder (RCA) or use a truncation protocol that "divides" a secret with a power of 2 [11,22,23,35,39,41]. Either of these approaches require round complexity logarithmic in the bit-size and is nearly optimal.

Definition of daBits: In an attempt to improve performance using a pre-processing model, Rotaru and Wood [32] recently proposed a primitive known as daBit. This provides an efficient solution to the problem of generating random bits (used for comparisons but also for other secure computation primitives). A daBit is a doubly authenticated bit:

$$\mathsf{daBit} := [b]_{2^\ell}, [b]_2 \quad \text{such that} \quad b \in \{0,1\} \tag{1}$$

Note that here, the $[\cdot]_.$ notation is used oblivious to the underlying adversarial model. In the case of honest majority corruption models, this simply refers to arithmetic shares of the secret. In the case of dishonest majority MPC where this primitive was first introduced [32], $[\cdot]_.$ refers to the additive sharing of the value and it's MAC. This primitive improved the performance of "inter-conversion" between arithmetic secret sharing and Boolean secret sharing, consequently improving the performance of secure comparisons. [32] used a cut-and-choose technique from literature [51,52] to generate many daBit with good concrete efficiency.

Definition of edaBits: Further improving upon [32], Escudero *et. al.* [22] proposed a primitive known as edaBit (extended doubly authenticated bit) that generates shares of a random value and it's bit decomposition. More formally, an edaBit is:

$$\begin{gathered}\mathsf{edaBit} := [b]_{2^\ell}, [b_0]_2, \cdots, [b_\ell - 1]_2 \\ \text{such that} \quad b \equiv \sum_{i=0}^{\ell-1} b_i 2^i \quad \text{and} \quad b_i \xleftarrow{\$} \{0,1\}\end{gathered} \tag{2}$$

Once again, the notation $[\cdot]_.$ is oblivious to the adversarial model and will refer to simple arithmetic shares for honest majority corruption models. In the case of dishonest majority [22], it will refer to arithmetic shares of the secret as well as it's MAC. Note that it is easy to generate an edaBit using ℓ-daBits by simply locally combining them. However, [22] demonstrated that it is possible to generate large number of edaBit with better efficiency than generating more daBits. It is important to note that for generating edaBits, the protocols that require the least amount of run-time use $O(\log_2 \ell)$ rounds of communication for generating this pre-processing material (my using secure adder circuits). In this work, we showcase a constant round protocol for generating edaBits that achieves lower run-time costs than prior state-of-the-art protocols.

Note that while the primitives daBit and edaBits were initially proposed in the dishonest majority setting, the primitives themselves are more general and can be considered in other adversarial models such as honest majority. For instance, [37] uses edaBits in a semi-honest setting for applications to secure federated learning. Note that when we compare the performance of prior state-of-the-art in Section 5, we benchmark the implementation with the same adversarial model as ours.

2.2. Silent OT Extensions

In our work, we make black-box use of silent OT extensions (in our implementation, we use a PCG-based construction for silent OT extensions [15]). Such OT correlations are generated in batches – a small amount of interactive work is performed to generate keys which are then expanded into a large batch of OT correlations. The idea, as outlined in the previous section, is divided into two components:

(1) *Generation*: The Gen algorithm takes the security parameter as input and generates two keys (k_0, k_1).
(2) *Expansion*: Given the keys generated by the Gen algorithm, the Expand algorithm outputs a vector of OT correlations, i.e., $(w_0[i], w_1[i])$ at the sender and $(u[i], v[i])$ where $i \in [n]$ and $v[i] = w_{u[i]}[i]$.

OT correlation functionality

Parameters: $1^\lambda, \ell, n \in \mathbb{N}$

(1) For $i \in [n]$, generate random values $w_0[i], w_1[i] \in \mathbb{F}_{2^\ell}$ and random bits $u[i] \in \mathbb{F}_2$. Define $v[i]$ as

$$v[i] = \begin{cases} w_0[i] & \text{if } u[i] = 0 \\ w_1[i] & \text{if } u[i] = 1 \end{cases}$$

(2) Send $(w_0[i], w_1[i])$ for $i \in [n]$ to the sender.
(3) Send $(u[i], v[i])$ for $i \in [n]$ to the receiver.

Figure 2: Functionality for OT correlation generation.

3-Party Oblivious Transfer Functionality $\mathcal{F}_{\mathsf{3OT}}$

$\mathcal{F}_{\mathsf{3OT}}$ runs with parties P_0, P_1, P_2 and an adversary $\mathcal{S}$ and proceeds as follows:

(1) $\mathcal{F}_{\mathsf{3OT}}$ receives (s_i^0, s_i^1, b_i) from party P_i for $i \in \{0, 1, 2\}$
(2) $\mathcal{F}_{\mathsf{3OT}}$ sends party P_i the value $s_{i-1}^{b_i}$ for $i \in \{0, 1, 2\}$

Figure 3: 3-party oblivious transfer functionality description.

This is captured by the functionality specified in Fig. 2. The protocols for securely emulating the above functionality is presented in Appendix A using the silent OT extension protocols from [15]. It is important to note that we make black-box use of the above functionality. Thus, our protocol can directly benefit from any improvement to the underlying silent OT extension protocol used to emulate this functionality. For instance, recent work [53] reduces the computational costs by 19× compared to [15] and these improvements will similarly affect the performance of our protocols.

3. Semi-honest Secure Protocol

In this section, we present protocols that are secure against a semi-honest adversary. In Section 4, we show how to extend these protocols to be maliciously secure. We argue for the security in the Arithmetic Black Box model $\mathcal{F}_{\mathsf{ABB}}$ (see for instance [22, 32, 54]), an ideal functionality based on the UC paradigm [55].

Notation. We use a replicated secret sharing scheme, used in a number of prior works sharing this adversarial model [25, 35, 56, 57], as the basis for our protocols. Thus each data value x is secret shared with party P_i holding the pair of values (x_{i+1}, x_{i+2}) for $i \in \{0, 1, 2\}$ such that:

$$x \equiv x_0 + x_1 + x_2 \pmod{2^\ell}$$

We use the notation $x.\mathsf{first}$ to denote the first component of the share on any given party, i.e., to denote x_{i+1} if called on party P_i (and $x.\mathsf{second}$ respectively to denote the second component). We use the terms *next party* and *previous party* in reference to a given party P_i to denote the party P_{i+1}, P_{i-1} respectively. These implicitly assume that the indices are taken modulo 3, thus party P_{i-1} when $i = 0$ is P_2 and P_{i+1} when $i = 2$ is P_0. We also refer to rounds as *clockwise* when each party P_i sends data to the next party and *counter-clockwise* when each party sends data to the previous party. Let $\mathcal{F}_{\mathsf{cr}}^1$ and $\mathcal{F}_{\mathsf{cr}}^2$ be the correlated randomness generation functionalities as given below:

(1) $\mathcal{F}_{\mathsf{cr}}^1$ generates random values $\alpha_0, \alpha_1, \alpha_2$ where P_i gets α_i such that $\alpha_0 + \alpha_1 + \alpha_2 \equiv 0 \pmod{2^\ell}$
(2) $\mathcal{F}_{\mathsf{cr}}^2$ generates random values $\alpha_0, \alpha_1, \alpha_2$ where P_i gets $(\alpha_{i+1}, \alpha_{i+2})$ such that $\alpha_0 + \alpha_1 + \alpha_2 \equiv 0 \pmod{2^\ell}$

For more details on securely emulating these functionalities, refer to Protocol 2.5 and Functionality 2.6 from [57]. Note that both these functionalities can be implemented without interaction assuming pairwise randomness and can be trivially extended to generate correlated randomness (of either type of sharing) for any given public value (not just zero).

We use pairwise PCGs for OT correlation set-up once during an initialization phase, i.e., PCG set-up is run once between each pair of parties such that the next party is always the Receiver of the OT and the previous party is the Sender[1]. We use the notation $\mathsf{k}_{P_i \to P_{i+1}}, \mathsf{k}_{P_{i+1} \leftarrow P_i}$ to denote the PCG keys at the P_i (the Sender) and P_{i+1} (the Receiver) respectively[2]. Note that these keys enable the two parties, *to non-interactively generate an OT correlation* – one sender party generating $(w_0, w_1) \in \mathbb{F}_2^\ell \times \mathbb{F}_2^\ell$ and the receiver party generating $(u, v) \in \mathbb{F}_2 \times \mathbb{F}_2^\ell$ such that $v = w_u$ (cf Section 2.2 for details). Before we present our semi-honest protocol for daBit generation, we introduce a 3-Party Oblivious Transfer (3OT) building block.

[1] Note that for each pair of parties, there is a unique party that is next for the other party and similarly for previous.

[2] The first subscript denotes which party holds the key, the arrow denotes it's role as a Sender/Receiver, and the second subscript indicates the other party in this pairwise OT correlation.

3-Party Oblivious Transfer Protocol Π_{3OT}

Inputs: Parties P_i for $i \in \{0,1,2\}$ each hold values $s_i^0, s_i^1 \in \mathbb{Z}_{2^\ell}$. Furthermore, each party also holds a bit $b_i \in \{0,1\}$.
Outputs: Each party P_i receives an output $s_{i-1}^{b_i}$
Common Randomness: For each $i \in \{0,1,2\}$, parties P_i, P_{i+1} hold OT Extension PCG keys where P_i acts as the Sender and P_{i+1} as the Receiver viz., $\mathsf{k}_{P_i \to P_{i+1}}, \mathsf{k}_{P_{i+1} \leftarrow P_i}$ respectively. Note that Step 1 of the protocol produces a large batch of OT correlations which are amortized across a batch of Π_{3OT} computation.

Protocol: For each $i \in \{0,1,2\}$, parties P_i and P_{i+1} run the following protocol pairwise:

(1) Run Π_{Expand} with the following set-up:
   (a) P_{i+1} on inputs $\sigma = 0, k_\sigma = \mathsf{k}_{P_{i+1} \leftarrow P_i}$ to get output (u, v).
   (b) P_i on inputs $\sigma = 1, k_\sigma = \mathsf{k}_{P_i \to P_{i+1}}$ to get output (w_0, w_1).

(2) P_{i+1} sends the bit $d = b_{i+1} \oplus u$ to P_i.

(3) P_i sends $(V^0, V^1) = (s_i^0 + w_d, s_i^1 + w_{1-d})$ to P_{i+1} and P_{i+1} outputs $V^{b_{i+1}} - v \in \mathbb{Z}_{2^\ell}$. Here $\mathbb{F}_2^\ell$ elements are interpret as $\mathbb{Z}_{2^\ell}$ for the addition and subtraction.

Figure 4: Protocol for secure 3-party oblivious transfer.

3.1. 3-Party Oblivious Transfer

In a 3-party oblivious transfer primitive, each party simultaneously acts as a Sender and a Receiver. Each party holds a choice bit b_i which will be used to select one of two messages (s_{i-1}^0, s_{i-1}^1) held by the previous party. The functionality is described in Fig. 3 and the protocol securely emulating this functionality is described in Fig. 4.

Construction. In a one time set-up, we run three PCGs for OT correlations, once between each pair of parties. For $i \in \{0,1,2\}$, when the protocol is run between P_i, P_{i+1}, P_i is the OT Sender and P_{i+1} is the Receiver. Thus each party P_i holds two keys, once as a Sender $\mathsf{k}_{P_i \to P_{i+1}}$ and once as a Receiver $\mathsf{k}_{P_i \leftarrow P_{i-1}}$. The protocol follows the simple structure of emulating an oblivious transfer using random oblivious transfer [31]. The Receivers mask their bit using the pre-processing oblivious transfer and send it to the Senders. The Senders masks it's values s_i^0, s_i^1 with the pre-processing oblivious transfer strings, in an order that depends on the masked bit revealed to it. This ensures that the Receiver can only unmask one of the two strings depending on it's secret bit. Note that recent PCG constructions [14, 15, 58, 59] all operate in batches and we account for this overhead in Section 5.

Correctness of Π_{3OT}. The correctness follows from the following series of equations:

$$\begin{aligned} V^{b_{i+1}} - v &= s_i^{b_{i+1}} + w_{b_{i+1} \oplus d} - w_u \\ &= s_i^{b_{i+1}} + w_u - w_u \\ &= s_i^{b_{i+1}} \end{aligned} \tag{3}$$

Where the first equations follows from the observations that $w_u = v$ (from the correctness of the OT Extensions of EXPAND sub-routine) and V^b can be succinctly written as $s_i^b + w_{b \oplus d}$. Thus, the output of party P_{i+1} is $s_i^{b_{i+1}}$ as expected.

Security of Π_{3OT}. The security of the protocol can be stated in the following theorem:

Theorem 3.1. *Protocol Π_{3OT} (Fig. 4) securely evaluates functionality $\mathcal{F}_{\mathsf{3OT}}$ (Fig. 3) with abort in the $\mathcal{F}_{\mathsf{PPRF-GGM}}$-Hybrid model in the presence of one maliciously corrupt party ($\mathcal{F}_{\mathsf{PPRF-GGM}}$ and constructions are defined Appendix A).*

Proof. Here we use the notation $x_{i,j}$ to denote a value sent from party $i \to j$. Let us assume without loss of generality that party P_0 is corrupt. We know that Π_{3OT} runs over 2 rounds, the first round involves a counter-clockwise sending of the bits d (Step (2) of Fig. 4) and the second round involves a clockwise sending of the values (V^0, V^1) (Step (3) of Fig. 4). Thus, the entire transcript of party P_0 is as follows: In round one, it receives one bit $d_{0,1}$ from party P_1 and sends a bit $d_{0,2}$ to party P_2. Similarly, in the second round, P_0 receives two values $(V_{0,2}^0, V_{0,2}^1)$ from party P_2 and sends two values $(V_{0,1}^0, V_{0,1}^1)$. Thus, we construct the simulator $\mathcal{S}$ as follows:

(1) Simulator generates a random bit $d_{0,1}$ and sends it to P_0. Simultaneously, the simulator receives the bit $d_{0,2}$ from party P_0 and extracts the input $b_0 = d_{0,2} \oplus u$ where u can be generated using simulation access to $\mathcal{F}_{\mathsf{PPRF-GGM}}$. Given that distributions of d are uniform over $\mathbb{Z}_2$, this perfectly simulates round 1 for the adversary.

(2) To simulate the second round, simulator generates the values $(V_{0,2}^0, V_{0,2}^1)$ honestly and sends them to P_0. It also receives two values $(V_{0,1}^0, V_{0,1}^1)$ from P_0 and extracts the values s_0^0, s_0^1 using simulation access to $\mathcal{F}_{\mathsf{PPRF-GGM}}$. It forwards these values to

the functionality $\mathcal{F}_{\mathsf{3OT}}$ and the honest parties receive their output. Note that the distribution of $V^0_{0,2}, V^1_{0,2}$ is uniformly random over $\mathbb{Z}_{2^\ell}$ and thus perfectly simulates the second round to the adversary.

The final check is that the honest parties receive outputs consistent with the functionality which holds because of the simulation extraction. □

Semi-honest daBit Generation $\Pi_{\mathsf{sh:daBit}}$

Inputs: Number N of daBits to be generated. Auxiliary parameters B, C.
Outputs: Parties share replicated secret sharings of a random bit b, i.e., $[b]_2, [b]_{2^\ell}$.

Protocol: All parties have access to functionality $\mathcal{F}_{\mathsf{3OT}}$.

(1) *Generate Random Sharings:* Here parties locally generate random secret shares required for Step (2).
   (a) Locally generate 3-out-of-3 shares of 0 and 1 in $\mathbb{Z}_{2^\ell}$ viz., (x_0, x_1, x_2) and (y_0, y_1, y_2) respectively, where P_i holds share $*_i$ for $i \in \{0,1,2\}$ and $* \in \{x, y\}$.
   (b) Locally generate replicated secret sharing of a random bit $[b]_2$ and 3-out-of-3 sharing of $[\alpha]_{2^\ell}$ where $\alpha = 0$ and P_i holds the shares (b_{i+1}, b_{i+2}) and α_i.

(2) *Oblivious Transfer:* Here, parties get 2-out-of-3 *arithmetic* sharings of the value b. Each party P_i computes the following:
   (a) If $b_{i+1} \oplus b_{i+2} = 0$ set $s^0 \leftarrow x_i$ and $s^1 \leftarrow y_i$
   (b) If $b_{i+1} \oplus b_{i+2} = 1$ set $s^0 \leftarrow y_i$ and $s^1 \leftarrow x_i$
   (c) Parties run Π_{3OT} with each party P_i calling the functionality inputs s^0, s^1 (when acting as the Sender) and b_{i+2} (when acting as the Receiver). The output received by P_i is stored as B_{i-1} $(= B_{i+2})$.
   (d) For each $i \in \{0,1,2\}$, party P_i computes $c_{i+2} \equiv B_{i+2} + \alpha_i \pmod{2^\ell}$ and sets it as $c.\mathsf{second}$). Furthermore, it sends this value c_{i+2} to party P_{i+1}. Finally, each party sets the received value as $c.\mathsf{first}$.

Figure 5: Protocol for *semi-honest secure* daBit generation.

Complexity of Π_{3OT}. The protocol runs in 2 rounds, the first round involving a single bit of communication and the second round involving a single element of $\mathbb{Z}_{2^\ell}$ per party. Thus the total communication overhead of this protocol is $(\ell + 1)$-bits per party, split over two rounds.

3.2. Semi-honest daBit Generation

We present an extremely simple protocol for daBit using the above described 3-party OT primitive. The protocol is described in Fig. 5.

Construction. The protocol is conceptually very simple, parties start with 3-out-of-3 secret sharing of 0 and 1. These serve as the arithmetic shares. Furthermore, they also hold a 2-out-of-3 secret sharing of a bit b which will serve as the selector bit. The central idea of the construction is to use the bit to select the appropriate sharing of either 0 or 1 (using the 3-party OT primitive) using the shares of the selection bit b. Note that we already have access to a 2-out-of-3 secret sharing of the bit. Thus we only need to generate a 2-out-of-3 arithmetic secret sharing of the same bit. Our protocol achieves this by having the parties use the selector bit to select shares of 0 or 1 and then use a re-sharing protocol to get a 2-out-of-3 secret sharing of the same bit b but as arithmetic shares.

More concretely, each party sets its string inputs s^0_i, s^1_i to be their own share of 0 or 1, depending on the value of the bit $b_{i+1} \oplus b_{i+2}$. In other words, if $b_{i+1} \oplus b_{i+2} = 0$, then the parties set s^d_i to be the their own arithmetic share of d for $d \in \{0,1\}$. The correctness of Π_{3OT} ensures that all parties receive the share corresponding to the value $b_i \oplus b_{i+1} \oplus b_{i+2}$. Note that this simple construction uses only 3 fixed rounds and is highly parallelizable.

Correctness of $\Pi_{\mathsf{sh:daBit}}$. Correctness follows easily from the following series of observations. First, each OT sends a share $*_i$ where $i \in \{0,1,2\}$ and $* \in \{x, y\}$. Second, given the ordering of s_0, s_1 set by each party, the output of each call to $\mathcal{F}_{\mathsf{OT}}$ is the same for each party and is perfectly correlated with the value $b_i \oplus b_{i+1} \oplus b_{i+2} = b$. Third, considering the boundary conditions, we see that if $b = 0$ then $* = x$ and if $b = 1$ then $* = y$. Finally, given each party sends its shares to the other two parties, at the end of the oblivious transfers, each party has 2-out-of-3 shares of either 0 or 1 (x or y depending on the bit b). More succintly, at the end of step 2(c), party P_i's output is given by:

$$\begin{aligned} \mathsf{Output} &= s^{b_{i+2}}_{i-1} \\ &= [b_{i+2} \oplus b_{i-1+1} \oplus b_{i-1+2}]_{\mathbb{Z}_{2^\ell}} \\ &= [b_i \oplus b_{i+1} \oplus b_{i+2}]_{\mathbb{Z}_{2^\ell}} = [b]_{\mathbb{Z}_{2^\ell}} \end{aligned} \tag{4}$$

where the second step follows from the fact that the conditions on step 2(a), 2(b), s^d holds arithmetic shares of $d = b_{i+1} \oplus b_{i+2}$. Finally, matching indices, it follows that

the parties get shares corresponding from the inputs held by the previous party and thus $\mathsf{Output} := B_{i+2}$.

Security of $\Pi_{\mathsf{sh:daBit}}$. The security of $\Pi_{\mathsf{sh:daBit}}$ can be easily proved in a hybrid model that assume access $\mathcal{F}_{\mathsf{3OT}}$ as well as two ideal functionalities that non-interactively generate 3-out-of-3 and 2-out-of-3 secret sharings of a constant, viz., $\mathcal{F}^1_{\mathsf{cr}}$, $\mathcal{F}^2_{\mathsf{cr}}$ respectively (refer to [57] for a formal treatment). The security of $\Pi_{\mathsf{sh:daBit}}$ can be stated as:

Theorem 3.2. *$\Pi_{\mathsf{sh:daBit}}$ securely generates a* daBit *in the ($\mathcal{F}_{\mathsf{3OT}}$, $\mathcal{F}^1_{\mathsf{cr}}$, $\mathcal{F}^2_{\mathsf{cr}}$)-hybrid model with abort in the presence of one semi-honest corruption (Functionalities defined in Fig. 3 and Fig. 12).*

Proof. Once again, let us assume without loss of generality that P_0 is the corrupt party. The simulator runs an internal copy of the protocol making use of $\mathcal{F}_{\mathsf{3OT}}$. The input generation can be replaced by calls to the the functionalities $\mathcal{F}^1_{\mathsf{cr}}$ and $\mathcal{F}^2_{\mathsf{cr}}$. The simulator, who controls $\mathcal{F}_{\mathsf{3OT}}$ can then extract the input $b_1 \oplus b_2$ provided by the adversary which can be fed into the ideal functionality. In both the worlds, the transcripts c_{i+2} are uniformly distributed in $\mathbb{Z}_{2^\ell}$ and subject to the output constraint of $\mathcal{F}_{\mathsf{3OT}}$. This establishes the indistinguishability of the transcripts as well as the output. □

Complexity of $\Pi_{\mathsf{sh:daBit}}$. The semi-honest protocol runs in 3 rounds – two rounds used by Π_{3OT} and a round used by the resharing. The resharing involves communication of a single element of $\mathbb{Z}_{2^\ell}$ per party. Thus the total communication overhead of this protocol is $(2\ell + 1)$-bits per party, split over three rounds.

4. Maliciously Secure Protocol

In this section, we extend the semi-honest protocol to be secure against malicious corruptions. We use a technique from literature known as cut-and-choose [51, 52] that is used to detect malicious behaviour with high statistical probability. While our work is follows the argument from [57], we note that improved cut-and-choose bounds such as those in [60] can reduce the overhead of achieving malicious security.

Construction. Our construction relies on the two key insights – the structure of the semi-honest protocol and the fact that there are two honest parties. The latter is exploited by ensuring that the protocols run in the same direction (clockwise). In other words, the construction ensures that the outputs of party P_{i+1} are controlled/influenced by party P_i but not by P_{i+2}. This ensures that at least one honest party generates the right output. Note that the semi-honest protocol does not guarantee consistency of the shares nor correctness of the daBits under malicious corruptions.

To address the final issue, since each individual daBit can be incorrect, we use a standard cut-and-choose technique from literature to generate many correct daBits with high confidence. The idea is to generate many such daBits, randomly shuffle them, open first few values to check for correctness. If they are all correct, the remaining values are put into a number of buckets and within each bucket, the first value is used to verify the remaining values. The parameters ($C :=$ how many daBits are opened and $B :=$ the bucketing size) are set so that such a protocol when successfully completed ensures all daBits are correct with high probability.

The protocol is formally described in Fig. 6. The first part of the protocol is to simply generate $M = (NB+C)$ daBits using the semi-honest protocol $\Pi_{\mathsf{sh:daBit}}$, for given parameters B, C and where N is the number of daBits desired. Then, the parties run a verification to check the consistency of their shares. This can be done using a collision resistant hash function. The semi-honestly generated daBits are then shuffled using a permutation functionality $\mathcal{F}_{\mathsf{perm}}$ (described in Appendix C.2). The shuffle can be efficiently performed in MPC by generating the seed within the MPC and then using a pseudorandom generator. The first C daBits are opened and checked for correctness and the remaining are grouped into N bins with B daBits per bin. The first daBit in each bin is verified using every other daBit in the bin using the subroutine Π_{verify}. If these complete successfully, output the first daBit of each bin.

Correctness of $\Pi_{\mathsf{mal:daBit}}$. The correctness of $\Pi_{\mathsf{mal:daBit}}$ follows from the correctness of $\Pi_{\mathsf{sh:daBit}}$. Before we prove the security of the protocol, we prove a lemma for Π_{verify} used for verifying one daBit using another.

Lemma 4.1. *If $[b]_2$ and $[c]_{2^\ell}$ is a valid* daBit *pair (i.e., both sharings are consistent and $b = c$) and $[b']_2$ and $[c']_{2^\ell}$ are consistent shares, with $b' \neq c'$, then all honest parties output $\perp$ in Π_{Verify}.*

Proof. We begin by observing that b, b', c, c' are all consistent shares. Thus, b, b' are shares of some bits, c is an arithmetic share of the same bit b, and c' is an arithmetic share of some value $\in \mathbb{Z}_{2^\ell}$. For a given value of the bit c, we note that the function $f_c : \mathbb{Z}_{2^\ell} \to \mathbb{Z}_{2^\ell}$ given by $f_c(c') = c + c' - 2 \cdot c \cdot c'$ forms is a bijection.

Malicious daBits Generation $\Pi_{\mathsf{mal:daBit}}$

Input: Number N of daBits to be generated. Auxiliary parameters B, C.
Output: Parties share replicated secret sharings of a random bit b, i.e., $[b]_2, [b]_{2^\ell}$.
Protocol:

(1) *Generate Random Sharings:* Here parties locally generate random sharings of 0 and 1.
   (a) Locally generate 3-out-of-3 shares of 0 and 1 in $\mathbb{Z}_{2^\ell}$ viz., (x_0, x_1, x_2) and (y_0, y_1, y_2) respectively, where P_i holds share $*_i$ for $i \in \{0, 1, 2\}$ and $* \in \{x, y\}$. (The parties call $\mathcal{F}_{\mathsf{cr}}^1$ on inputs 0, 1 respectively)
   (b) Locally generate replicated secret sharing of a random bit $[b]_2$ and 3-out-of-3 sharing of $[\alpha]_{2^\ell}$ where $\alpha = 0$ and where P_i holds the share (b_{i+1}, b_{i+2}) and α_i. (The parties call $\mathcal{F}_{\mathsf{cr}}^2$ and $\mathcal{F}_{\mathsf{cr}}^1$ respectively)

(2) *Oblivious Transfer:* Here, parties get 2-out-of-3 *arithmetic* sharings of the value b. Each party P_i computes the following:
   (a) If $b_{i+1} \oplus b_{i+2} = 0$ set $s^0 \leftarrow x_i$ and $s^1 \leftarrow y_i$
   (b) If $b_{i+1} \oplus b_{i+2} = 1$ set $s^0 \leftarrow y_i$ and $s^1 \leftarrow x_i$
   (c) Parties run Π_{3OT} with each party P_i calling the functionality inputs s^0, s^1 (when participating as the Sender) and b_{i+2} (when participating as the Receiver). The output received by P_i is stored as B_{i-1} $(= B_{i+2})$.
   (d) For each $i \in \{0, 1, 2\}$, party P_i computes $c_{i+2} \equiv B_{i+2} + \alpha_i \pmod{2^\ell}$ and sets it as $c.\mathsf{second}$). Furthermore, it sends this value c_{i+2} to party P_{i+1}. Finally, each party sets the received value as $c.\mathsf{first}$.

(3) *Consistency check:* Let $M = NB + C$. Steps (1)-(2) are repeated M times to generate a vector $\mathbf{D}$ of size M. The parties then generate two hash values, the first on all the $c.\mathsf{first}$ and the second on all the $c.\mathsf{second}$ values. The parties commit and verify these values are consistent.

(4) *Cut and Choose:* Finally, the parties run the cut-and-choose protocol to produce N correct daBits.
   (a) Call $\mathcal{F}_{\mathsf{perm}}$ on the vector $\mathbf{D}$ i.e., on the vector of shares $\{([b^i]_2, [c^i]_{2^\ell})\}$ for $i \in \{1, 2, \cdots, M\}$.
   (b) Open the first C values of the vector M. If any of the reconstructions fail or if $b^i \neq c^i$, send abort to the other parties and output $\perp$.
   (c) Remove the first C values from the vector $\mathbf{D}$ and divide the rest of the shares into N buckets of size B.
   (d) For each bucket, verify the correctness of the first element (shares of b^i, c^i) using all the other daBits in that bucket by running the subroutine Π_{Verify}.
   (e) If all verifications succeed and no party outputs abort, output the first daBit from each bucket.

Figure 6: Protocol for *maliciously secure* daBit generation.

In fact, it is an involution which can be seen from:

$$\begin{aligned} f_c(f_c(c')) &= f_c(c + c' - 2 \cdot c \cdot c') \\ &= c + (c + c' - 2 \cdot c \cdot c') \\ &\qquad - 2 \cdot c \cdot (c + c' - 2 \cdot c \cdot c') \\ &= (2c - 2c^2) + c' - 4c'(-c + c^2) = c' \end{aligned} \tag{5}$$

Where the last equation uses the fact that $c \in \{0, 1\}$. Thus, there exists a unique value of c' that satisfies $f_c(c') = b \oplus b'$. However, we know that $b' \in \mathbb{Z}_{2^\ell}$ satisfies $f_c(b') = b \oplus b'$ as $c = b$. Hence, the check ensures that either the honest parties abort in the multiplication in Π_{verify} or $b' = c'$. This completes the proof.

Note that Step (2)(d) ensures that c forms consistent shares of some value in $\mathbb{Z}_{2^\ell}$. We will use this along with the cut and choose technique to ensure that c forms a consistent share of b or the honest parties abort the protocol in some intermediate stage.

Security of $\Pi_{\mathsf{mal:daBit}}$. The security of $\Pi_{\mathsf{mal:daBit}}$ can be formally stated as follows:

Theorem 4.2. *Let $B = C = 3$ and let $N \geq 2^{\mathsf{sec_s}/2}$ for some statistical security paramter $\mathsf{sec_s}$. Then, $\Pi_{\mathsf{mal:daBit}}$ securely generates a N daBits in the ($\mathcal{F}_{\mathsf{3OT}}$, $\mathcal{F}_{\mathsf{perm}}$, $\mathcal{F}_{\mathsf{cr}}^1$, $\mathcal{F}_{\mathsf{cr}}^2$)-hybrid model, with statistical security $2^{-\mathsf{sec_s}}$ in the presence of one malicious corruption.*

The simulator for this follows the same lines as the simulator for the semi-honest case. Crucially, the Simulator $\mathcal{S}$ sends continue to the ideal functionality only if the semi-honest protocol, i.e., Steps 1,2, and 3 succeed. The only difference between the real execution and the ideal execution is when the cut-and-choose protocol succeeds over incorrect inputs. Thus, the security reduces to showing that the probability that an adversary can cheat in the cut-and-choose game and evade detection is bounded above by $2^{-\mathsf{sec_s}}$. The analysis of this problem is considered in [52] and improved in [51]. However, in this work, we use the bounds from [57] which improve upon both these works[3]. Below we prove this bound.

[3]Another recent work [60] improves these bounds further by leveraging their application scenario.

Verifying one daBit using Another Π_{Verify}

Inputs: Parties hold one valid daBit pair $[b]_2, [c]_{2^\ell}$ and another pair $[b']_2, [c']_{2^\ell}$.
Outputs: Parties output accept if $[b']_2, [c']_{2^\ell}$ is a valid daBit pair and abort otherwise.

Protocol: The verification works as follows:
(1) Compute $[d]_2 = [b \oplus b']_2$
(2) Compute $[e]_{2^\ell} = [c \oplus c']_{2^\ell}$. Note that this is computed as the expression $(c + c' - 2 \cdot c \cdot c')$ over 2^ℓ.
(3) Open d and e. If $d = e$, output accept else output abort.

Figure 7: Protocol for verifying one daBit using another daBit without opening.

Proof. Given the consistency check in Step 3 of $\Pi_{\mathsf{mal:daBit}}$, we know that the adversary can only force the inputs to Step 4 (cut-and-choose section) to be consistent sharings of non-conforming values, i.e., the bit b and the value c do not agree. Thus, each daBit can be tagged as bad if it is incorrect and good if it is correct. We can then model the success probability of the adversary by simply analyzing the probability as a function of the number of bad daBits. Note that the adversary succeeds only if the following two conditions hold:

(1) No bad daBit is opened among the C opened.
(2) There is no mixed-binning, i.e., there is no bin with both good and bad daBits.

where the probability is over the random permutation. Here the first conditions leads to abort from the honest parties and the second one holds because Π_{verify} will abort on any mixed-bin. Suppose that the adversary corrupts $T = B \cdot t$ number of daBits[4] where $t \geq 1$. Let $\mathcal{E}_1$ be the event that condition 1 holds and $\mathcal{E}_2$ the event that condition 2 holds (here, the C daBits and the binning is fixed after the permutation).

$$\Pr[\mathcal{E}_1] = \frac{\binom{M-T}{C}}{\binom{M}{C}} \tag{6}$$

The probability that all the T bad daBits fall into t bins over a random permutation can be computed as follows: we pick the t bins that contain all the bad daBits and then look at the probability that the specific set of bad daBits is chosen to fit into these bins. Thus, there are $\binom{N}{t}$ ways to select the bad bins and the probability that our exact set of bad daBits is chosen to fit in them is just 1 in the total number of ways to select T objects from NB objects. Thus, the probability of event $\mathcal{E}_2$ happening given event $\mathcal{E}_1$ has happened is[5]:

$$\Pr[\mathcal{E}_2|\mathcal{E}_1] = \frac{\binom{N}{t}}{\binom{NB}{T}} \tag{7}$$

Note that the success probability of our adversary is exactly captured by the event $\mathcal{E}_1 \wedge \mathcal{E}_2$. Thus, we can compute it as:

$$\begin{aligned}\Pr[\mathcal{E}_2 \wedge \mathcal{E}_1] &= \Pr[\mathcal{E}_2|\mathcal{E}_1] \cdot \Pr[\mathcal{E}_1] \\ &= \frac{\binom{N}{t}\binom{M-T}{C}}{\binom{NB}{T}\binom{M}{C}} = \frac{\binom{N}{t}}{\binom{M}{T}}\end{aligned} \tag{8}$$

Restricting the parameter regime to $C \geq B$, we can bound this probability by:

$$\Pr[\mathcal{E}_2 \wedge \mathcal{E}_1] = \frac{\binom{N}{t}}{\binom{M}{T}} \leq \frac{\binom{N}{t}}{\binom{NB+B}{Bt}} \tag{9}$$

Let us define the last expression by $f_{N,B}(t)$. We note the following two properties of this function:

(1) If $t \leq \lfloor N/2 \rceil$, then $f_{N,B}(t) \geq f_{N,B}(N-t)$.
(2) If $t \leq \lfloor N/2 \rceil$, then $f_{N,B}(t-1) \geq f_{N,B}(t)$

To prove the first statement, we first note that if $2t = N$, then the statement holds trivially since both the expressions are the same. Suppose $2t \leq N-1$. We note that the numerators of the two function evaluations are equal, i.e., $\binom{N}{t} = \binom{N}{N-t}$. However, the denominator is strictly larger in the second evaluation as:

$$\begin{aligned}\binom{NB+B}{B(N-t)} &= \binom{NB+B}{BN-Bt} \\ &= \binom{NB+B}{NB+B-(BN-Bt)} \\ &= \binom{NB+B}{Bt+B} \\ &\geq \binom{NB+B}{Bt}\end{aligned} \tag{10}$$

Where the second equality follows from the symmetry of the binomial coefficients and the last expression follows

[4]The second condition implies that T has to be a multiple of B for the adversary to succeed.

[5]Another way to see this is to pick t bins out of the N, permute the T bad daBits in them and the $(NB-T)$ good daBits in the remaining bins and then divide this by the total number of ways permuting NB daBits.

from the monotonicity of the binomial coefficients and the fact that $2(Bt+B) \leq (N-1)B + 2B = NB + B$.

To prove the second statement, we consider the following binomial identity relating adjacent coefficients:

$$\binom{n}{k} = \frac{n+1-k}{k}\binom{n}{k-1} \tag{11}$$

We need to show that $f_{N,B}(t-1)/f_{N,B}(t) \geq 1$. Repeated use of the above identity in the expression leads to:

$$\begin{aligned}\frac{f_{N,B}(t-1)}{f_{N,B}(t)} &= \frac{t}{N+1-t} \cdot \frac{\binom{NB+B}{Bt}}{\binom{NB+B}{Bt-B}} \\ &= \frac{t}{N+1-t} \cdot \frac{(NB+B+1-Bt)}{Bt} \cdot \\ &\quad \frac{(NB+B+2-Bt)}{Bt-1} \cdots \frac{(NB+B+B-Bt)}{(Bt-B+1)}\end{aligned} \tag{12}$$

Here the second term $(NB+B+1-Bt)/Bt < (NB+B-Bt)/Bt = (N+1-t)/t$ and thus the product of the first two terms is greater than 1. We argue that each of the remaining fractions are also ≥ 1. Indeed, for $i \in \{1, 2, \cdots, B-1\}$,

$$\begin{aligned} & \frac{NB+B+1-(Bt-i)}{Bt-i} && \geq 1 \\ \iff & NB+B+1-(Bt-i) && \geq Bt-i \\ \iff & NB+B+1 && \geq 2Bt-2i \end{aligned}$$

However, the maximum value of $2Bt-2i$ in the range for t is $(N-1)B-2$. This proves the two claims. Combining these two claims, we know that the maximum value of $f_{N,B}(t)$ is at $t=1$. Thus,

$$\begin{aligned}\Pr[\mathsf{Adv.\ Success}] &= \Pr[\mathcal{E}_2 \wedge \mathcal{E}_1] \\ &\leq N \cdot \binom{NB+B}{B}^{-1}\end{aligned} \tag{13}$$

Typical value of statistical security used in literature is $\mathsf{sec_s} = 40$. Setting $B = 2$, we get that the failure probability scales roughly as $O(1/N)$ and thus requires large batches of about 2^{40}. Using $B = 3$, the adversarial success probability is bounded above by $1/4N^2$. Thus, setting $B = 3, N \geq 2^{\mathsf{sec_s}/2}$ would suffice[6]. □

Complexity of $\Pi_{\mathsf{mal:daBit}}$. The malicious protocol build over the semi-honest protocol. We consider the amortized cost to produce N daBits. The semi-honest protocol runs over 3 rounds and uses $M(2\ell+1)$-bits of communication. Assuming the consistency check is performed using a hash function over λ-bits, the communication incurred for that is 3λ-bits per party split over two rounds (commitment can be combined into λ-bits and opening requires 2λ-bits). Furthermore, the seed generation of the permutation also takes $O(\lambda)$-bits of communication over two rounds. However, both these costs are independent of N and hence will be ignored in the estimates. Opening C daBits will require $(\ell+1)$-bits per party and the verification is performed using $N(B-1)(2\ell+2)$-bits of communication over two rounds (for all the bins).

Thus the total cost for the entire protocol is equal to $M(2\ell+1) + O(\lambda) + N(B-1)(4\ell+4)$-bits split over 10 rounds. Substituting $M = NB + C$ and using $N \approx 2^{20}$ (which gives $2^{-\mathsf{sec_s}} = 2^{-40}$) and $B = C = 3$, we get the overall amortized communication complexity of $\Pi_{\mathsf{mal:daBit}}$ is roughly $14\ell + 11$. For values of ℓ used in practice, 32 or 64-bits, this is roughly 32 and 64 Bytes respectively (14 ring elements).

5. Evaluation

We evaluate the efficiency of our protocols in comparison to the state of the art frameworks in both the semi-honest and honest majority frameworks. More specifically, we measure the time taken to generate a million daBits and a million edaBits. We consider both the semi-honest and malicious settings as well as both the LAN and WAN network settings.

Experimental Set-up. We run over experiments over set-up similar to a number of prior works [25, 34, 61]. Our machines use Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz processors with 32 cores. For the LAN setting, our machines have a bandwidth of 10Gbps and have an average RTT of 0.23ms. For the WAN setting, the bandwidth is capped to 200Mbps with 80ms of RTT. We set number of threads to 32 to parallelize our protocols and set the parameters $B = C = 3$ for the cut-and-choose. For comparison with prior work, we run the state-of-the-art implementations over MP-SPDZ [38] to run benchmark on the set-up as our work. *Note that we compare the protocols from MP-SDPZ with the appropriate threat model (semi-honest/malicious and honest majority corruption).* Both our work and prior work are full implementations of the primitives and not microbenchmarks.

[6] The more exact expression would be $N \geq 2^{\mathsf{sec_s}-2/2}$.

5.1. Protocols for daBit Generation

We evaluate the cost taken to generate a million daBits using our protocol. These reuslts are presented in Tables 1, 2. For prior work, we benchmark the generation of daBit generation from the state-of-the-art implementation of [32] from [41] over the same set-up as our work. We use the get_dabit() function call run using replicated-ring-party.x executable for the semi-honest setting and malicious-rep-ring-party.x for the malicious run. While our protocol requires slightly higher communication in comparison to [32], we achieve about a 10× lower overhead and thus an 10× higher throughput in the LAN setting. We can generate about 23 million daBits per second. In the WAN setting, this is reduced to about 3 million daBits per second, about 5× improvement over prior work.

When considering the malicious setting, our protocol provides only marginally better throughput over LAN. The reason is that our protocol requires an expensive local computation – shuffling which forms the dominant cost of the protocol. However, these improvements are more pronounced when considering a network constrained environments such as WAN. In this setting, our protocol achieves about 9× higher throughput in comparison to prior art. These gains result from the low round complexity of our protocol in comparison to prior art.

Protocol	64-bit Semi-honest daBits		
	Time (LAN)	Time (WAN)	Comm.
This Work	0.0428	0.3435	16.125 MB
MP-SPDZ [38]	0.4277	16.4331	13.362 MB

Table 1: Time required (in seconds) for the generation of a million daBits using the semi-honest protocol from Section 3.

.

Protocol	64-bit Maliciously secure daBits		
	Time (LAN)	Time (WAN)	Comm.
This Work	3.0164	9.361	518 MB
MP-SPDZ [38]	3.6698	78.1039	51.028 MB

Table 2: Time (in seconds) for the generation of a million daBits using the maliciously secure protocol from Section 4.

.

5.2. Protocols for edaBit Generation

Similar to the comparison for daBit generations, for prior work, we benchmark the generation of edaBits using the state-of-the-art implementation of [32] from [41] over the same set-up as our work. We use the get_edabit() function call run using replicated-ring-party.x executable for the semi-honest setting and malicious-rep-ring-party.x for the malicious run. Our protocols when used for edaBit generation provide more modest gains. This is because we use a naive technique of combining multiple daBits to generate one edaBit that results in high round complexity over a batch (refer to Section 6.1 for improved concrete efficiency). Table 3 shows the performance of our semi-honest edaBit generation in comparison to [22] while Table 4 in the malicious setting. Note that these timings are for 64-bit data types.

Protocol	64-bit Semi-honest edaBits		
	Time (LAN)	Time (WAN)	Comm.
This Work	1.63658	10.9521	1032 MB
MP-SPDZ [38]	5.91341	355.628	45.487 MB

Table 3: Time required (in seconds) for the generation of a million edaBits using the semi-honest protocol from Section 3.

Protocol	64-bit Maliciously secure edaBits		
	Time (LAN)	Time (WAN)	Comm.
This Work	193.0505	599.042	33.152 GB
MP-SPDZ [38]	48.2178	971.121	820.97 MB

Table 4: Time (in seconds) for the generation of a million edaBits using the maliciously secure protocol from Section 4.

In semi-honest security, our protocols achieve about 4× higher throughput in the LAN setting which improves to about 32× in the WAN setting. We can thus produce about 100k edaBits per seconds. However, when considering the malicious setting, we note that [22] outperforms our protocols in the LAN setting. The reason for this is that in our protocol, we have to generate maliciously secure daBits and then simply locally combine them. The reason for this is the difficulty of constructing a Π_{Verify} protocol (similar to the one in Fig. 7) for verifying one edaBit using another without

opening. Such a protocol can be constructed using an n-bit adder protocol and we present more details in Section 6. Such protocols will require round complexity proportional to logarithm of the bit length but will yield much higher throughput as this would require generating only semi-honest daBits. In the WAN setting, our maliciously secure protocols already outperform prior art due to the low round complexity.

5.3. Other Experimentals

We also run experiments to benchmark the performance of our protocol when considering smaller data types such as 32-bit values, when considering a more optimized malicious edaBit generation protocol (described in Section 6), and the key generation costs.

32-bit data-types. While it is common to use 64-bit data-types for secure computation, in some applications, it is sufficient to use 32-bit data-types [25]. Thus, we also benchmark our protocols for generation of 32-bit daBit and edaBit. We note that communication overhead is simply proportional to the bit-size and thus reduces by a factor of 2 for daBit generation and 4× for edaBits (due to the quadratic dependence of the bit-size ℓ). When implemented, semi-honest daBit generation throughput increases by 1.89× while the throughput of edaBit generation improves by 3.78× and can generate over 1.38 million edaBits per second. In the WAN setting, the throughput of daBits improves by 1.32× and that of edaBits improves by 2.64×. These results are summarized in Table 5.

Protocol	32-bit Semi-honest generation		
	Time (LAN)	Time (WAN)	Comm.
daBits	0.0225	0.2595	8.0625 MB
edaBits	0.7216	8.304832	258 MB

Table 5: Time required (in seconds) and communication for the generation of a million daBits and edaBits for 32-bit data-types using the semi-honest protocol from Section 3.

Efficient Malicious edaBits. As described in Section 6, a more concretely efficient protocol for malicious edaBit generation can be obtained by applying the cut-and-choose techniques directly over semi-honest edaBits. The procedure to generate such edaBits is very similar to that described in Fig. 6 except with the difference that semi-honest daBits will be generated to obtain a semi-honest edaBits and then the cut-and-choose subroutine will guarantee correctness with high probability.

More specifically, to generate N maliciously secure edaBits, we first generate $(3N+3)\ell$ semi-honest daBits, locally combine them to form $(3N+3)$ semi-honest edaBits. We then run the cut-and-choose protocol using the Π_{Verify2} routine (refer to Fig. 8) to verify the first edaBit in each bucket against the rest. In terms of concrete efficiency, this brings down the overhead of edaBit generation from 193.05 seconds to 12.11 seconds in the LAN setting and from 599.04 seconds to 106.07 seconds in the WAN setting.

Key generation costs. We also run other microbenchmarks to measure the performance of our protocols. The cost of producing all the silent OT extension keys required for a million daBits is 0.416 seconds for the semi-honest case and the 1.248 seconds for the malicious case. Furthermore, note that is a batched cost and thus is generated one-time.

6. Discussion

In this section, we describe how the concrete efficiency of the malicious edaBit generation protocol can be improved, extension of the protocols to secure computation over fields, as well as some open questions in this space.

6.1. Improved Malicious edaBit Generation

The protocol for maliciously secure edaBit generation presented in Section 4 follows a simple structure – to generate N edaBits of a given bit-length ℓ, generate ℓN maliciously secure daBits and then simply locally combine them. As each maliciously secure daBit is expensive to generate, we can improve the concrete efficiency at the cost of an increased round complexity.

The challenge of deferring the cut-and-choose is that for such an approach, we would require a protocol to verify one edaBit using another edaBit without opening. While there is a simple 2 round protocol for a similar problem for daBits, the simplest construction for edaBits uses round complexity proportional to logarithm of the bit length. Such a protocol is defined in Fig. 8. The protocol uses an n-bit adder subroutine using only Boolean shares which can be highly efficient (refer to [22, 41] for a description of these primitives). The benefit of such an approach is that semi-honest edaBits (using semi-honest daBits) can be generated with about two orders

of magnitude higher throughput which further amortizes the expensive computational cost shuffling which is a dominant cost in the cut-and-choose approach.

6.2. Extension to Prime Moduli

Different MPC frameworks and protocols require different representations of the function computation and operate over different datatypes. When MPC protocols are implemented, the actual values to be computed upon can be integers or floating-point values (converted into fixed-point values) and are thus assumed to lie within some integer range $[-2^{k-1}, \cdots, 2^{k-1}]$. However, the choice of the modulus is an important consideration. These values can be operated on as values in a group $\mathbb{Z}_M = \{0, 1, 2, \cdots, M-1\}$ and all computation is performed as group operations. When the modulus $M = 2^\ell$ is a power of 2, where the resulting group is a ring. Computations over $\mathbb{Z}_{2^\ell}$ win in their handling of overflow in modular arithmetic. Thus a number of recent state-of-the-art MPC frameworks focus on the advancing the MPC protocol design in this setting [22, 33, 35, 62, 63].

Verifying one edaBit using Another Π_{Verify2}

Inputs: Parties hold $[x]_{2^\ell}, [x_0]_2, \cdots, [x_{\ell-1}]_2$, a valid edaBit and a set of shares $[y]_{2^\ell}, [y_0]_2, \cdots, [y_{\ell-1}]_2$.
Outputs: Parties output accept if $[y]_{2^\ell}, [y_0]_2, \cdots, [y_{\ell-1}]_2$ is a valid edaBit and abort otherwise.

Protocol: The verification works as follows:

(1) The parties invoke $\mathcal{F}_{\mathsf{ABB}}$ to compute $\mathsf{nBitADD}(x_0, \cdots x_{\ell-1}; y_0, \cdots, y_{\ell-1})$ and obtain ℓ-bits $[z]_2, \cdots [z_{\ell-1}]_2$ (ignoring the carry bit).
(2) Call ConvertB2A from $\mathcal{F}_{\mathsf{ABB}}$ to convert $[z_i]_2 \to [w_i]_{2^\ell}$.
(3) Parties open

$$a = \sum_{i=0}^{\ell-1} 2^i \, [w_i]_{2^\ell}$$

and $b \equiv x + y \pmod{2^\ell}$. If $a = b$, output accept else output abort.

Figure 8: Protocol for verifying one edaBit using another edaBit without opening.

On the contrary, when the modulus M is a prime, the resulting group is a field and thus enjoys stronger properties such as the existence of inverse or high probability detection of malicious behaviour (proportional to the field size). The protocols presented in Section 3, 4 can both be extended to work over fields. The protocols only require a way to generate random shares of 0,1 over fields and protocols for functionalities $\mathcal{F}_{\mathsf{cr}}^1$ and $\mathcal{F}_{\mathsf{cr}}^2$ are easy to extend to fields. In the case of edaBit generation, a small caveat exists that the space of elements in the field (say $\mathbb{Z}_p$) is smaller than the space of $\lceil \log p \rceil$ bits. However, contrary to the approach of [22], this is not an issue in our construction as many daBits $[b]_p, [b]_2$ can be locally combined to generate a correct edaBit. The maliciously secure protocol with higher concrete efficiency (described in Section 6.1) is used, then the protocol requires that the circuit for addition, i.e., the n-bit adder used in the protocol be performed modulo p.

6.3. Limitations and Open Questions

While our protocols increase the through of daBit and edaBit generation by about an order of magnitude, they incur a higher communication overhead. Furthermore, in our protocol, edaBits are generated naively – by combining daBits equal to the bit-length of the data types. This is known to be sub-optimal as shown in [22]. However, given the construction of our protocol, it is non-trivial to combine the insights of [22] to produce edaBits more efficiently while achieving this constant round complexity.

An important open question is to see if the techniques from [22] be used to improve the performance of edaBit generation. In particular, can we do better than generating bit-size number of daBits for each edaBit? Another interesting open question is to consider extending these techniques to 2PC or general nPC settings – these protocols crucially rely on the properties of replicated secret sharing in a 3-party computation model. It is also unclear if these constructions can be extended to dishonest majority adversarial models.

7. Related Work

There has been a lot of progress in recent years in efficient secure computation protocols in the pre-processing model. Below we describe some of the recent works relevant to our contributions.

PCG constructions. Pseudorandom Correlation Generators (PCGs) allow generation of correlated randomness using short seeds. Oblivious Transfers and OT Extensions are a common type of correlation that has been widely studied. Recent advances [14, 15, 58, 59] have provided constructions for efficient "silent" OT

extensions. Other works such as [60,64,65] also focus on efficient generation of VOLE correlations or OT correlations with low interaction. These advances build upon seminal works in function secret sharing [66–68]. A number of other works use the advances in FSS schemes to construct improved MPC protocols [69–72]. The techniques from this work can improve other applications limited by the performance of cryptography [73–76]. Being compute dominated, all these works can further benefit from hardware acceleration implementation platforms such as Piranha [77] and CryptGPU [78].

Secure comparison. A number of works study the generation of bits in arithmetic secret sharing, a primitive which is a critical component of many MPC frameworks [13, 62, 79]. These primitives form the basis of most pre-processing required for secure comparisons. The notion of doubly authenticated bits aka daBits was proposed in a seminal work by Rotaru and Wood [32]. This idea was extended to maBits in [80] which providing sharing across different frameworks. These works allow efficient share conversions as described in [81]. More recently, the daBit primitive was extended to extended doubly authenticated bits aka edaBit in [22].

Efficient comparison protocols have been studied almost since the inception of the field of secure computation. Recent comparison protocols in the 2-party setting include [21, 82, 83], and in more general n-party setting [9–12]. Works such as [25, 33, 84] propose secure comparison protocols in fixed 3PC/2PC models. More recently, Makri *et. al.* [23] propose highly efficient comparison protocols in general n-party model in the arithmetic black box model. Their work builds upon the edaBit primitive and currently forms the state-of-the-art online protocol for general n-party computation.

Other frameworks. A number of recent works have considered the 3-party computation model for secure computation. The seminal works [56,57] proposed protocols for semi-honest and malicious adversaries that have been the foundation for a number of follow-up works. Works such as [25,33] provide more efficient comparison protocols. Another suit of works also improve the efficiency of protocols using hybrid conversions [34, 35, 85]. A few frameworks consider a 4-party set-up and achieve stronger properties such as guaranteed output delivery or fairness [44, 61, 85].

Other frameworks such as CryptFlow, CryptFlow2 [84, 86] improve upon prior work in adversarial model and performance. Recent work SIRNN [50] focuses on providing a mathematical library for commonly used functions in secure computation. FantasticFour [44] provides a new approach for achieving malicious security in a 4PC setting. ABY [40] was a framework in the 2PC setting that provided efficient conversions between arithmetic, Boolean, Yao's GC representations enabling efficient hybrid protocols. ABY^3 [35] provided extensions of this idea to the 3PC setting and more recently ABY2.0 [87] provides further improvements in the 2PC setting. Another work [36] studies applications of PPML to quantized neural networks. Another work [18] improve function dependent pre-processing and improves upon the SPDZ line of work [13, 79].

8. Conclusion

We propose new protocols for generating daBits and edaBits in the 3-party computation model. These build upon a 3-party oblivious transfer functionality and enjoy a constant round complexity using advances in silent pseudorandom correlation generators. Our implementation shows that these new protocols improve the state-of-the-art by an order of magnitude.

References

[1] A. Yao, "Protocols for Secure Computations," in *Foundations of Computer Science (FOCS)*, 1982.

[2] A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1986.

[3] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *ACM Symposium on Theory of Computing (STOC)*, 1987.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *ACM Symposium on Theory of Computing (STOC)*, 1988.

[5] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols (extended abstract)," in *ACM STOC*, 1988.

[6] Z. Galil, S. Haber, and M. Yung, "Cryptographic computation: Secure faut-tolerant protocols and the public-key model," in *Advances in Cryptology—CRYPTO*, 1987.

[7] D. Chaum, I. Damgård, and J. van de Graaf, "Multiparty computations ensuring privacy of each party's

input and correctness of the result," in *Advances in Cryptology—CRYPTO*, 1987.

[8] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay – A secure two-party computation system," in *USENIX Security Symposium (USENIX)*, 2004.

[9] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally Secure Constant-Rounds Multi-Party Computation for Equality, Comparison, Bits and Exponentiation," in *Theory of Cryptography Conference (TCC)*. Springer, 2006, pp. 285–304.

[10] T. Nishide and K. Ohta, "Multiparty Computation for Interval, Equality, and Comparison without Bit-Decomposition Protocol," in *International Workshop on Public Key Cryptography*. Springer, 2007, pp. 343–360.

[11] O. Catrina and S. De Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks*, 2010.

[12] H. Lipmaa and T. Toft, "Secure Equality and Greater-Than Tests with Sublinear Online Complexity," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2013, pp. 645–656.

[13] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[14] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent ot extension and more," in *Advances in Cryptology—CRYPTO*, 2019, pp. 489–518.

[15] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round ot extension and silent non-interactive secure computation," in *ACM Conference on Computer and Communications Security (CCS)*, 2019, pp. 291–308.

[16] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *USENIX Security Symposium (USENIX)*, 2020.

[17] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A Low Latency Framework for Secure Neural Network Inference," in *USENIX Security Symposium (USENIX)*, 2018, pp. 1651–1669.

[18] A. B. Efraim, M. Nielsen, and E. Omri, "Turbospeedz: Double your online spdz! improving spdz using function dependent preprocessing," in *Applied Cryptography and Network Security (ACNS)*, 2019.

[19] S. Wagh, "New Directions in Efficient Privacy Preserving Machine Learning," Ph.D. dissertation, Princeton University, 2020.

[20] H. Chen, M. Kim, I. Razenshteyn, D. Rotaru, Y. Song, and S. Wagh, "Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning," in *Advances in Cryptology—ASIACRYPT*, 2020.

[21] G. Couteau, "New Protocols for Secure Equality Test and Comparison," in *Applied Cryptography and Network Security (ACNS)*, 2018.

[22] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, "Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits," in *Advances in Cryptology—CRYPTO*, 2020.

[23] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh, "Rabbit: Efficient Comparison for Secure Multi-Party Computation," in *Financial Cryptography and Data Security (FC)*, 2021.

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

[25] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning," in *Privacy Enhancing Technologies Symposium (PETS)*, 2021.

[26] M. O. Rabin, "How to exchange secrets with oblivious transfer," Harvard University Technical Report. http://eprint.iacr.org/2005/187.

[27] J. Killian, "Founding crytpography on oblivious transfer," in *ACM Symposium on Theory of Computing (STOC)*, 1988.

[28] Y. Ishai, M. Prabhakaran, and A. Sahai, "Founding cryptography on oblivious transfer – efficiently," in *Advances in Cryptology—CRYPTO*, 2008.

[29] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Advances in Cryptology—CRYPTO*, D. Boneh, Ed., 2003.

[30] V. Kolesnikov and R. Kumaresan, "Improved ot extension for transferring short secrets," in *Advances in Cryptology—CRYPTO*, R. Canetti and J. A. Garay, Eds., 2013.

[31] M. Keller, E. Orsini, and P. Scholl, "Actively secure ot extension with optimal overhead," in *Advances in Cryptology—CRYPTO*, 2015.

[32] D. Rotaru and T. Wood, "Marbled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security," in *International Conference on Cryptology in India*. Springer, 2019, pp. 227–249.

[33] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-Party Secure Computation for Neural Network Training," in *Privacy Enhancing Technologies Symposium (PETS)*, 2019.

[34] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "Astra: High throughput 3pc over rings with application to secure prediction," in *ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2019.

[35] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[36] D. Escudero, A. Dalskov, and M. Keller, "Secure evaluation of quantized neural networks," in *Privacy Enhancing Technologies Symposium (PETS)*, 2020.

[37] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy preserving aggregate statistics via boolean shares," https://eprint.iacr.org/2021/576, 2021.

[38] Data61, "MP-SPDZ: Versatile Framework for Multi-party Computation," https://github.com/data61/MP-SPDZ, 2019.

[39] A. Aly, M. Keller, E. Orsini, D. Rotaru, P. Scholl, N. P. Smart, and T. Wood, "SCALE-MAMBA v1.2: Documentation," https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf, 2018.

[40] D. Demmler, T. Schneider, and M. Zohner, "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation," in *Symposium on Network and Distributed System Security (NDSS)*, 2015.

[41] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," in *ACM Conference on Computer and Communications Security (CCS)*, 2020.

[42] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," in *USENIX Security Symposium (USENIX)*, 2021.

[43] S. Carpov, K. Deforth, N. Gama, M. Georgieva, D. Jetchev, J. Katz, I. Leontiadis, M. Mohammadi, A. Sae-Tang, and M. Vuille, "Manticore: Efficient framework for scalable secure multiparty computation protocols," Cryptology ePrint Archive, Report 2021/200, 2021, https://eprint.iacr.org/2021/200.

[44] A. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *USENIX Security Symposium (USENIX)*, 2021.

[45] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[46] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[47] C. Baum, L. Braun, A. Munch-Hansen, and P. Scholl, "Appenzeller to brie: Efficient zero-knowledge proofs for mixed-mode arithmetic and Z_{2^k}," 2021, https://eprint.iacr.org/2021/750.

[48] C. Baum, B. David, and T. Frederiksen, "P2dex: Privacy-preserving decentralized cryptocurrency exchange," in *Applied Cryptography and Network Security (ACNS)*, 2021.

[49] D. J. Wu, T. Feng, M. Naehrig, and K. Lauter, "Privately evaluating decision trees and random forests," in *Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[50] D. Rathee, M. Rathee, R. K. K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "Sirnn: A math library for secure rnn inference," in *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[51] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A new approach to practical active-secure two-party computation," in *Advances in Cryptology—CRYPTO*, R. Safavi-Naini and R. Canetti, Eds., 2012.

[52] S. S. Burra, E. Larraia, J. B. Nielsen, P. S. Nordholt, C. Orlandi, E. Orsini, P. Scholl, and N. P. Smart, "High performance multi-party computation for binary circuits based on oblivious transfer," *IACR Cryptology ePrint Archive*, 2015, https://eprint.iacr.org/2015/472.

[53] G. Couteau, P. Rindal, and S. Raghuraman, "Silver: Silent vole and oblivious transfer from hardness of decoding structured ldpc codes," in *Advances in Cryptology—CRYPTO*, 2021.

[54] I. Damgård and J. B. Nielsen, "Universally composable efficient multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology—CRYPTO*. Springer, 2003, pp. 247–264.

[55] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001.

[56] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[57] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, "High-throughput secure three-party computation for malicious adversaries and an honest majority," in *Advances in Cryptology—EUROCRYPT*, 2017.

[58] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast Extension for coRRElated oT with small communication," in *ACM Conference on Computer and Communications Security (CCS)*, 2020, pp. 1607–1626.

[59] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova, "Distributed vector-ole: improved constructions and implementation," in *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[60] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits," in *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[61] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "FLASH: Fast and robust framework for privacy-preserving machine learning," in *Privacy Enhancing Technologies Symposium (PETS)*, 2020.

[62] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "SPDZ2k: Efficient MPC mod 2^k for Dishonest Majority," in *Advances in Cryptology—CRYPTO*, 2018.

[63] P. Mohassel and Y. Zhang, "SecureML: A System for Scalable Privacy-Preserving Machine Learning," in *IEEE Symposium on Security and Privacy (S&P)*, 2017.

[64] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Correlated pseudorandom functions from variable-density lpn," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2020.

[65] ——, "Efficient pseudorandom correlation generators from ring-lpn," in *Advances in Cryptology—CRYPTO*, 2020.

[66] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Advances in Cryptology—EUROCRYPT*, 2015.

[67] ——, "Function secret sharing: Improvements and extensions," in *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[68] J. Doerner and abhi shelat, "Scaling oram for secure computation," in *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[69] E. Boyle, N. Gilboa, and Y. Ishai, "Secure computation with preprocessing via function secret sharing," in *Theory of Cryptography Conference (TCC)*, 2019.

[70] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing vector ole," in *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[71] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Advances in Cryptology—EUROCRYPT*, 2021.

[72] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," in *Privacy Enhancing Technologies Symposium (PETS)*, 2022.

[73] S. Wagh, X. He, A. Machanavajjhala, and P. Mittal, "DP-Cryptography: marrying differential privacy and cryptography in emerging applications," 2020.

[74] S. Wagh, P. Cuff, and P. Mittal, "Differentially private oblivious RAM," in *Privacy Enhancing Technologies Symposium (PETS)*, 2018.

[75] M. Costa, L. Esswood, O. Ohrimenko, F. Schuster, and S. Wagh, "The Pyramid scheme: Oblivious RAM for trusted processors," in *Tech Report*, 2017, https://arxiv.org/abs/1712.07882.

[76] D. M. Sommer, L. Song, S. Wagh, and P. Mittal, "Towards probabilistic verification of machine unlearning," in *Under submission*, 2020.

[77] J.-L. Watson, S. Wagh, and R. Ada Popa, "Piranha: A GPU Platform for Secure Computation," in *USENIX Security Symposium (USENIX)*, 2022.

[78] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the GPU," in *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[79] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical Covertly Secure MPC for Dishonest Majority–or: Breaking the SPDZ Limits," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 1–18.

[80] D. Rotaru, N. P. Smart, T. Tanguy, F. Vercauteren, and T. Wood, "Actively Secure Setup for SPDZ," Cryptology ePrint Archive, Report 2019/1300, 2019, https://eprint.iacr.org/2019/1300.

[81] A. Aly, E. Orsini, D. Rotaru, N. P. Smart, and T. Wood, "Zaphod: Efficiently combining lsss and garbled circuits in scale," in *ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019.

[82] T. Toft, "Sub-Linear, Secure Comparison with Two Non-Colluding Parties," in *International Workshop on Public Key Cryptography*. Springer, 2011, pp. 174–191.

[83] C.-H. Yu and B.-Y. Yang, "Probabilistically Correct Secure Arithmetic Computation for Modular Conversion, Zero Test, Comparison, MOD and Exponentiation," in *International Conference on Security and Cryptography for Networks*. Springer, 2012, pp. 426–444.

[84] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[85] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "Swift: Super-fast and robust privacy-preserving machine learning," in *USENIX Security Symposium (USENIX)*, 2021.

[86] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *ACM Conference*

on Computer and Communications Security (CCS), 2020.

[87] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation," in *USENIX Security Symposium (USENIX)*, 2021.

[88] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM (JACM)*, 1986.

[89] D. Boneh and B. Waters, "Constrained pseudorandom functions and their applications," in *Advances in Cryptology—ASIACRYPT*, 2013.

[90] E. Boyle, S. Goldwasser, and I. Ivan, "Functional signatures and pseudorandom functions," in *Public Key Cryptography (PKC)*, 2014.

A. Silent OT Correlations

Here we describe the protocol used to silently generate OT correlations in detail. We make black-box use of this functionality to generate OT correlations between parties. It is sufficient for the protocols proposed in this work to note that two parties can non-interactively call the protocol Π_{Expand} to expand the OT correlation seeds to produce a large number (say 10^7) of OTs.

(1) Key generation phase: The Gen algorithm takes the security parameter as input and generates two keys (k_0, k_1). The protocol used to compute these keys inside 2PC is given in Fig. 10.
(2) Expansion phase: Given the key k_σ for party σ ($\sigma = 0$ is the OT receiver and $\sigma = 1$ is the OT sender), the Expand algorithm outputs a vector of OT correlations corresponding to the sender/receiver role. The protocol used to compute these keys inside 2PC is given in Fig. 11.

Gen uses $\mathcal{F}_{\mathsf{OT}}$ which can emulated using any 2-round semi-honest OT protocol [15,29,31]. Each pair of parties run the above routine to generate a large batch of OT correlations (refer to Sec. 3 for details). During the 3-party oblivious transfer routine, the parties run Π_{Expand} to receive this batch of OT correlations. In this section, we present the details of the Gen and Expand routine for completeness and refer the reader to [15] for further details.

Notation. We use $[n]$ to denote the set $\{1, 2, \cdots, n\}$ for a given positive integer n, boldface letters such as $\mathbf{x}, \mathbf{y}$ to denote vectors, and use $y[i]$ to denote the i^{th} component of a vector $\mathbf{y}$. α_i is used to denote the i^{th} bit of a value α. Let $t, N = 2^k, n = N/s, \ell$ be parameters of the protocols, where $t \approx 120$ for the dual-LPN assumption, $N \approx 10^7$, s is a small constant such as 2, and ℓ is the modulus of the secret sharing scheme, set to 32 or 64.

The protocols use t-puncturable pseudorandom function (t-PPRF) that contains three algorithms – KeyGen, Puncture, and Eval. PPRF can be constructed from any length-doubling pseudorandom generator using the GGM tree-based construction [88–90]. The tree-evaluation takes a key k and point x (size of the tree is $n = \log|x|$), sets $k^0 \leftarrow k$ and performs the the following evaluation iteratively: for $i = 1$ to n, compute $(k_0^i, k_1^i) \leftarrow G(k^{i-1})$ and set $k^i \leftarrow k_{x_i}^i$ where x_i is the i^{th} bit of x. The procedure outputs k^n.

(1) KeyGen samples a random key from the key space.
(2) Puncture takes a key k and point x, performs the above procedure and sets $k\{x\} = \{k_{1-x_i}^i\}_i$ for $i \in [n]$. This protocol is presented in Fig. 10.
(3) Eval on input a punctured key $k\{x\}$ and a point x' outputs if $x = x'$. Otherwise, $k\{x\}$ is parsed as $\{k_{1-x_i}^i\}_i$ and start the iterative process at $k_{1-x_i}^i$ such that $x_i' = 1 - x_i$

For the LPN assumption, the t evaluations are XORed to give secret shared vectors or can be concatenate them with the regular LPN assumption. The two parties run an interactive protocol Π_{Gen} that makes use of any OT protocol [15,29,31] over a small number t of GGM trees. These tree evaluations can then be used to combined to satisfy the LPN assumption thus giving the parties secret shares of punctured pseudorandom function. The parties can non-interactively expand these keys using the Π_{Expand} protocol to generate a large batch of OT correlations. Finally, we make black box use of such a functionality to generate OT correlations that form the basis of our 3-party OT protocol Π_{3OT}. The ideal functionality $\mathcal{F}_{\mathsf{PPRF-GGM}}$ is given in Fig. 9.

Ideal Functionality $\mathcal{F}_{\mathsf{PPRF-GGM}}$

Input: Sender inputs $\beta \in \mathbb{F}_2^\ell$ and a PPRF key $k_{\mathsf{pprf}} \in \{0,1\}^\lambda$. Receiver inputs $\alpha \in \{0,1\}^\ell$.

Functionality:
(1) Compute $k_{\mathsf{pprf}}^* = \mathsf{PPRF.Puncture}(k_{\mathsf{pprf}}, \alpha)$
(2) Send k_{pprf}^* and $t = \beta - \mathsf{PPRF.Eval}(k_{\mathsf{pprf}}, \alpha)$ to Receiver.

Figure 9: Ideal functionality for distributing PRG correlation.

Seed Generation for OT Correlations Π_{Gen}

Input: Party P_0 (receiver) starts with a weight t vector $\mathbf{e} \in \mathbb{F}_2^N$. Let S be the location of the t non-zero indices i.e., $S = \{\alpha_1, \cdots, \alpha_t\}$. Party P_1 (sender) starts with a random value $x \in \mathbb{F}_{2^\ell}$ and t-PPRF keys $\{k_{\mathsf{pprf}}[i]\}$ for $i \in [t]$.
Output: Party P_0 (receiver) outputs the key $k_0 = (\{k^*_{\mathsf{pprf}}[i]\}_i, S, \{z_i\}_i)$ for $i \in [t]$. Party P_1 (sender) outputs the key $k_1 = (\{k_{\mathsf{pprf}}[i]\}_i, x)$ for $i \in [t]$.
Parameters: $N = 2^k, \ell \in \mathbb{N}$ and PPRF is a puncturable PRF with domain $[N]$ and range $\mathbb{F}_{2^\ell}$ constructed from the length doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ and a second PRG $G' : \{0,1\}^\lambda \to (\mathbb{F}_{2^\ell})^2$ for the last level of the GGM tree.

Protocol:
(1) Sender picks $\mathbf{b} \xleftarrow{\$} \mathbb{F}_{2^\ell}^t$ and send $\mathbf{c} = x - \mathbf{b}$ to the receiver.
(2) Run t-independent Puncture sub-routines for the t inputs $\alpha_i \in [N]$ and $b[i], k_{\mathsf{pprf}}[i]$ for $i \in [t]$. Receiver receives output $(k^*_{\mathsf{pprf}}[i], y_i)$ for $i \in [t]$
(3) Sender outputs $(\{k_{\mathsf{pprf}}[i]\}_i, x)$
(4) For $i \in [t]$ receiver computes $z_i \leftarrow y_i + c_i$.
(5) Receiver outputs $(\{k^*_{\mathsf{pprf}}[i]\}_i, S, \{z_i\}_i)$.

Puncture: Sub-routine used in the protocol. Sender inputs value $b \in \mathbb{F}_{2^\ell}$ and a seed k_{pprf} while receiver inputs a location $\alpha \in [N]$.
(1) Sender and Receiver execute ℓ calls in parallel to $\mathcal{F}_{\mathsf{OT}}$, where for $i = 1$ to ℓ-1:

- Receiver uses as input the choice bit α_i (i^{th} bit of α)
- Sender uses computes the 2^i partial evaluations at level i of the GGM tree defined by seed k_{pprf}, denoted by $s^i_0, \cdots, s^i_{2^i-1}$ and uses the following input to the OT:

$$t^i_L = \bigoplus_{j \in [0, 2^{i-1})} s^i_{2j} \quad \text{and} \quad t^i_R = \bigoplus_{j \in [0, 2^{i-1})} s^i_{2j+1}$$

For the last OT:

- Receiver uses as input the choice bit α_ℓ
- Sender uses computes the 2^ℓ evaluations of the GGM tree defined by seed k_{pprf}, denoted by $s^i_0, \cdots, s^i_{2^\ell-1}$ and uses the following input to the OT:

$$t^\ell_L = \sum_{j=0}^{2^{\ell-1}} s^i_{2j} \quad \text{and} \quad t^\ell_R = \sum_{j=0}^{2^{\ell-1}} s^i_{2j+1}$$

(2) In parallel to the OT calls, Sender sends $c = b - (t^\ell_L + t^\ell_R)$ to the receiver.

Receiver computes it's output as follows:
(1) Let t^1 be the Receiver's output in the first OT. Define $s^1_{\bar{\alpha_1}} = t^1$
(2) For $i = 2, ..., \ell$-1

- Compute $(s^i_{2j}, s^i_{2j+1}) = G(s^{i-1}_j)$ for $j \in [0, \cdots, 2^{i-1}), j \neq \alpha_1, \cdots, \alpha_{i-1}$
- Let t^i be the output from the i^{th} OT.
- Define $\alpha^*_i = \alpha_1, \cdots, \alpha_{i-1}, \bar{\alpha_i}$. Compute

$$s^i_{\alpha^*_i} = t^i \bigoplus_{j \in [0, 2^{i-1}), j \neq \alpha^*_i} s^i_{2j+\bar{\alpha_i}}$$

(3) Compute $(s^\ell_{2j}, s^\ell_{2j+1}) = G'(s^{\ell-1}_j)$ for $j \in [0, \cdots, 2^{\ell-1}), j \neq \alpha_1, \cdots, \alpha_{\ell-1}$. Receive c from the Sender and compute

$$y = c + t^\ell + \sum_{j=0, j \neq \alpha}^{2^{\ell-1}} s^i_{2j+\alpha_\ell}$$

(4) Receiver outputs the punctured key $k^*_{\mathsf{pprf}} = \{s^i_{\alpha^*_i}\}_{i \in [\ell]}$ and the final correction word y.

Figure 10: Protocol to run Gen phase of OT Correlations.

B. PCG Constructions

A Pseudorandom Correlation Generator (PCG) allows two (or more) parties to securely generate a large number of correlated randomness using short correlated seeds but only using local computation (thus requiring no interaction). Recently, a number of works have proposed novel constructions of PCGs for Oblivious Transfers [14, 15, 58, 59]. All these approaches rely on variants of the Learning Parity with Noise assumption (LPN) and advances in Function Secret Sharing (FSS), particularly schemes for Distributed Point Functions (DPF).

An FSS scheme allows multiple parties (say m parties) to additively split a function $f \to f_1, f_2, \cdots, f_m$ such that (1) each function is succinct (2) each strict subset of $\{f_1, \cdots, f_m\}$ hides f and (3) $f \equiv f_1 + f_2 + \cdots + f_m$ (equal at each point). A DPF scheme is one where the function f is a point function, i.e., is non-zero at exactly one input value. A closely related yet simpler primitive is a Puncturable Pseudorandom Function (PPRF) which is a PRF that allows evaluation at all points in the domain except at one point. The constructions

OT Correlation Expansion Protocol Π_{Expand}

Input: Party P_σ starts with key k_σ – output of Π_{Gen}.
Outputs: Party P_0 (receiver) gets a vector of size n of values $(u, v) \in F_2 \times \mathbb{F}_{2^\ell}$ and Party P_1 (sender) gets a vector of size n of values $(w_0, w_1) \in F_{2^\ell} \times \mathbb{F}_{2^\ell}$.
Parameters: $n, N, \ell \in \mathbb{N}$ with $N > n$. Matrix $H \in \mathbb{F}_2^{N \times n}$ and weight t error distribution over F_2^N (i.e., contains t non-zero indices with value 1). PPRF is a puncturable PRF with domain $[N]$ and range $\mathbb{F}_{2^\ell}$. And $\mathsf{Hash} : \{0,1\}^\lambda \times \mathbb{F}_{2^\ell} \to \{0,1\}^\lambda$ is a $\mathbb{F}_2$ correlation robust hash function.

Receiver protocol: Party $\sigma = 0$ acts as the receiver and runs the following protocols for extending OTs.
(1) Parse $k_0 = (\{k^*_{\mathsf{pprf}}[i]\}_i, S, \{z_i\}_i)$ for $i \in [t]$
(2) Define the error vector e using S.
(3) For $j \in [N]$ define the j^{th} entry of v_0 as

$$v_0[j] = \begin{cases} z_i & \text{if } \; j = \alpha_i \in S \\ \mathsf{PPRF.Eval}(k^*_{\mathsf{pprf}}[i], j) & \text{if } \; j \notin S \end{cases} \tag{14}$$

(4) Define $(\mathbf{u}, \mathbf{v'}) \leftarrow (\mathbf{e} \cdot H, -\mathbf{v_0} \cdot H)$
(5) For $i \in [n]$, define $v[i] = \mathsf{Hash}(i, v'[i])$.
(6) Output $(u[i], v[i])$ for $i \in [n]$.

Sender protocol: Party $\sigma = 1$ acts as the sender and runs the following protocols for extending OTs.
(1) Parse $k_1 = (\{k_{\mathsf{pprf}}[i]\}_i, x)$ for $i \in [t]$.
(2) Compute $\mathbf{v_{1i}} \leftarrow \mathsf{PPRF.FullEval}(k_{\mathsf{pprf}}[i]) \in \mathbb{F}_{2^\ell}^N$.
(3) Set $\mathbf{v_1} \leftarrow \oplus_i \mathbf{v_{1i}}$ and $\mathbf{w'} \leftarrow \mathbf{v_1} \cdot H$
(4) For $i \in [n]$, set

$$\begin{aligned} w_0[i] &\leftarrow \mathsf{Hash}(i, w'[i]) \\ w_1[i] &\leftarrow \mathsf{Hash}(i, w'[i] - x) \end{aligned} \tag{15}$$

(5) Output $(w_0[i], w_1[i])$ for $i \in [n]$.

Figure 11: Protocol for "silently" generating OT Correlations.

differ slightly in their approach and underlying cryptographic hardness assumption, here we will describe the approach from [15]. With each batch, their approach generates n set of correlated oblivious transfers (COT). This is achieved by first generating $N = 2n$ COTs with very high concrete efficiency and then applying a linear transformation to get n-pseudorandom COTs (this is the cryptographic hardness assumption, known as Dual LPN). Below we describe each step in more detail:

Multi-point Correlated Oblivious Transfer. In this phase, the Sender and Receiver together compute N-COTs

$$x_s + x_r = e \cdot \Delta \tag{16}$$

where the Sender gets $x_s \in \mathbb{F}_{2^\kappa}^N$ and $\Delta \in \mathbb{F}_{2^\kappa}$ and the receiver gets $x_s \in \mathbb{F}_{2^\kappa}^N$ and $e \in \mathbb{F}_2^N$. The trick is that the vector e is of low hamming weight, i.e., has only t-non-zero indices where t is a small constant. Suppose that $e = e_1 + e_2 + \cdots + e_t$ where each e_i is a vector of hamming weight one. Then Eq. 16 can be split into t equations such that:

$$x_{s,i} + x_{r,i} = e_i \cdot \Delta \tag{17}$$

Thus, viewing Eq. 17 as a DPF sharing, the entire set of N-COTs can be generated using only t-calls to an FSS scheme for DPFs. To further improve the performance, DPFs can be reduced by PPRFs and using the fact that the punctured location is known to the Receiver, the Sender simply sends the punctured key along with the "corrected" value at the punctured point.

Linear transformation. The second part of the construction is to use a linear transformation $H : \mathbb{F}^N \to \mathbb{F}^n$ where H is a parity check matrix of a linear code for which the LPN assumption holds. Each party locally applies this transformation to their shares, i.e.,

$$\begin{aligned} & y_s = Hx_s, \; y_r = Hx_r \;\; \text{and} \;\; e' = He \\ \Rightarrow \;\; & y_s + y_r = e' \cdot \Delta \end{aligned} \tag{18}$$

where e' is pseudorandom under the dual-LPN assumption. Using further optimizations that allow sampling a regular sparse noise vector e (where it is publicly known that exactly one index is set in each N/t-sized interval), the above construction allows generating n COTs using $O(t\kappa \log N/t)$-bits of communication.

Other constructions. Other constructions [58, 59] use a slightly different approach. They rely on the hardness of the LPN problem which changes the construction in subtle but important ways. To generate n-COTs, these approaches generate n-COTs x_s, x_r, e, Δ with sparse noise efficiently (similar to the multi-point COT above with n instead of N). The sparsity of the two approaches is slightly different (refer to [58, 59] for details). The parties also generate k COTs y_s, y_r, e', Δ where $k \approx \sqrt{n}$. Finally, let G to be the generator matrix of a linear code for which the LPN assumption holds. Then, the n-COTs are output using the linear transformations:

$$\begin{aligned} & z_s = x_s + Gy_s, \; z_r = y_r + Gx_r \;\; \text{and} \;\; e'' = e + Ge' \\ & \Rightarrow \;\; z_s + z_r = e'' \cdot \Delta \end{aligned} \tag{19}$$

Note that the communication of this approach $O((t \log n/t + k)\kappa)$-bits can be higher than that of the Dual-LPN based construction but the latter allows using simpler linear codes.

C. Other Ideal Functionalities

We present ideal functionalities for a random shuffle and the secure generation of correlated randomness.

C.1. Correlated Randomness - $\mathcal{F}_{\mathsf{cr}}^1/\mathcal{F}_{\mathsf{cr}}^2$

The ideal functionalities for the generation of correlated randomness are presented in Fig. 12. Note that these are straightforward generalizations of the Boolean functionalities presented in [57].

C.2. Random Shuffle - $\mathcal{F}_{\mathsf{perm}}$

In our protocol, we require random permutation of an array of elements (where each element is either a daBit or edaBit). $\mathcal{F}_{\mathsf{perm}}$ is an ideal functionality that receives a vector v of length M from all the parties, chooses a random permutation π over $\{1, 2, \cdots, M\}$, and returns a vector v' of the same length such that $v'[i] = v[\pi(i)]$. Note that because the permutation need not be secret from the parties and hence can be locally computed, such a functionality can be securely emulated by generating 128 random bits and then using a known PRG to generate randomness locally. Refer to [57] for more details.

Ideal Functionality $\mathcal{F}_{\mathsf{cr}}^1/\mathcal{F}_{\mathsf{cr}}^2$

Let $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \to D$ be a keyed function where the domain D is either $\{0,1\}$ or $\mathbb{Z}_{2^\ell}$. Upon invocation, the adversary controlling party P_i chooses a pair of keys $k, k' \in \{0,1\}^\kappa$ and sends them to $\mathcal{F}_{\mathsf{cr}}^1/\mathcal{F}_{\mathsf{cr}}^2$. Then:

Functionality $\mathcal{F}_{\mathsf{cr}}^1$:

(1) Upon receiving input id from all parties, functionality $\mathcal{F}_{\mathsf{cr}}^1$ computes $\alpha_i = F_k(\mathsf{id}) + F'_k(\mathsf{id})$ and chooses random values $\alpha_{i-1}, \alpha_{i+1} \in \{0,1\}$ under the constraint that $\alpha_0 + \alpha_1 + \alpha_2 = 0$ where the addition is done over D. $\mathcal{F}_{\mathsf{cr}}^1$ sends α_j to P_j for every j.

Functionality $\mathcal{F}_{\mathsf{cr}}^2$:

(1) Upon receiving input id from all parties, functionality $\mathcal{F}_{\mathsf{cr}}^2$ computes $\alpha_i = F_k(\mathsf{id})$ $\alpha_{i-1} = F'_k(\mathsf{id})$ and chooses a random value $\alpha_{i+1} \in D$. $\mathcal{F}_{\mathsf{cr}}^2$ sends $(\alpha_{j-1}, \alpha_j$ to P_j for every j.

Figure 12: Ideal functionality correlated randomness.