Tight Bounds on the Randomness Complexity of Secure Multiparty Computation

Vipul Goyal[1,2], Yuval Ishai[3], and Yifan Song[1]

[1] Carnegie Mellon University, USA.
vipul@cmu.edu, yifans2@andrew.cmu.edu
[2] NTT Research, USA.
[3] Technion, Israel.
yuvali@cs.technion.ac.il

Abstract. We revisit the question of minimizing the *randomness complexity* of protocols for secure multiparty computation (MPC) in the setting of perfect information-theoretic security. Kushilevitz and Mansour (*SIAM J. Discret. Math.*, 1997) studied the case of n-party semi-honest MPC for the XOR function with security threshold $t < n$, showing that $O(t^2 \log(n/t))$ random bits are sufficient and $\Omega(t)$ random bits are necessary. Their positive result was obtained via a non-explicit protocol, whose existence was proved using the probabilistic method.

We essentially close the question by proving an $\Omega(t^2)$ lower bound on the randomness complexity of XOR, matching the previous upper bound up to a logarithmic factor (or constant factor when $t = \Omega(n)$). We also obtain an *explicit* protocol that uses $O(t^2 \cdot \log^2 n)$ random bits, matching our lower bound up to a polylogarithmic factor. We extend these results from XOR to general *symmetric* Boolean functions and to addition over a finite Abelian group, showing how to amortize the randomness complexity over multiple additions.

Finally, combining our techniques with recent randomness-efficient constructions of private circuits, we obtain an explicit protocol for evaluating a general circuit C using only $O(t^2 \cdot \log |C|)$ random bits, by employing additional "helper parties" who do not contribute any inputs. This upper bound too matches our lower bound up to a logarithmic factor.

1 Introduction

The *randomness complexity* of probabilistic algorithms and distributed protocols is an important complexity measure that has been the subject of a large body of research. From a practical point of view, the design of algorithms and protocols that use a minimal amount of randomness is motivated by the difficulty of generating high-quality randomness from physical sources. While pseudorandomness provides a generic way of reducing the amount of randomness in a computational setting, this solution (besides requiring unproven cryptographic assumptions) is not always practical, especially in a distributed setting or when parties may be subject to resetting attacks. This motivated a line of work on minimizing the amount of randomness used by secure cryptographic hardware [24,22,2,3,17,14,19]. From a theoretical perspective, the goal of minimizing the use of randomness is a fundamental challenge that has driven many important developments in computer science, including a rich theory of pseudorandomness and randomness extraction.

This work studies the randomness complexity of *secure multiparty computation* (MPC) in the simplest setting of *perfect* security against a *passive* (semi-honest) adversary who may corrupt up to t parties. Such an MPC protocol allows n parties, each holding a local input $x_i \in D_i$, to jointly compute a function $f : D_1 \times D_2 \times \ldots \times D_n \to Z$ of their inputs by exchanging messages over secure point-to-point channels. At the end of the protocol, all parties should learn $f(x_1, x_2, \ldots, x_n)$. We say that the protocol is *t-secure* if every set of at most t parties jointly learn nothing beyond what follows from their inputs and the output. To achieve this goal, the parties may toss random coins at any time during the protocol's execution, possibly depending on their inputs and the messages they receive. The randomness complexity of the protocol is the total number of random bits used by all parties.

Classical MPC protocols for this setting [4,11] can compute every function f with randomness complexity $\tilde{O}(s \cdot t^2)$, where s is the Boolean circuit size of f, as long as $t < n/2$. (For bigger thresholds t, most functions cannot be realized at all in the information-theoretic setting.) In the useful special case of the XOR function, where $f(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ (or more generally, addition over a finite Abelian group), the "textbook" protocol from [5,13] requires $O(nt)$ random bits for any $t < n$.

The question of minimizing the randomness complexity of MPC has been the topic of a fairly large body of work [29,27,31,6,9,18,25,30,21,7,22,16,28]. While some of these works focus on the minimal security thresholds of $t = 1$ or $t = 2$, here we are interested in how the randomness complexity grows with t.

We will be mainly interested in the simple special case of computing the XOR function and, more generally, addition over finite Abelian groups, but will also consider other classes of functions f, including symmetric functions and even general functions. The case of addition is particularly well motivated because of its usefulness for many applications, including secure voting [5], anonymous communication [10], linear sketching [23], privacy-preserving analytics [15], federated learning [8], and more.

The randomness complexity of XOR was studied by Kushilevitz and Mansour [27], who proved that $O(t^2 \log(n/t))$ random bits are sufficient and $\Omega(t)$ random bits are necessary. This leaves a quadratic gap between the two bounds. Another question left open by [27] is the existence of an *explicit* protocol meeting the upper bound. The positive result was obtained via a non-explicit protocol, relying on a combinatorial object that can either be generated by an efficient probabilistic construction (with small but nonzero failure probability) or generated deterministically in super-polynomial time. Blundo et al. [6] obtain a lower bound of $\Omega(t^2/(n-t))$, which is asymptotically matched by the upper bound of [5,13] when $t = n - \Omega(1)$, but still leaves a quadratic gap when $t \leq (1-\epsilon)n$.

1.1 Our Contribution

In this work, we settle the main open questions about the randomness complexity of t-secure MPC for XOR and addition over finite Abelian groups, and obtain similar results for other functions. Concretely, we obtain the following results.

Lower bounds. We prove an $\Omega(t^2)$ lower bound on the randomness complexity of XOR, matching the previous upper bound of Kushilevitz and Mansour [27] up to a logarithmic factor (or even a constant factor when $t = \Omega(n)$). Our lower bound extends to arbitrary symmetric Boolean functions, including AND and majority. It applies also when the output is revealed to a strict subset of the parties and even in the case where there are additional participating parties who do not hold an input.

Our lower bounds do *not* apply to statistically private (let alone computationally private) MPC for the following inherent reason: in the setting of statistical privacy, one of the parties can pick a random committee $\mathcal{P}$ of σ parties, for a statistical security parameter σ, and the parties can securely add their inputs by secret-sharing them among the parties in $\mathcal{P}$. This folklore protocol, which is statistically $2^{-\Omega(\sigma)}$-secure against any (non-adaptive) adversary corrupting $t = 0.99n$ parties, has randomness complexity $O(n \cdot \sigma)$, which beats our $\Omega(n^2)$ lower bound when $\sigma = o(n)$. This explains the quick deterioration of the information-theoretic lower bound technique from [6], which is robust to small statistical deviations, when t gets farther away from n. Indeed, our lower bound proof relies on combinatorial rather than information-theoretic methods.

Explicit upper bounds for XOR and addition. To complement our lower bounds, we obtain an *explicit* protocol for XOR that uses $O(t^2 \cdot \log^2 n)$ random bits, matching our lower bound up to a polylogarithmic factor and at most a polylog-factor worse than the non-explicit protocol from [27]. We extend the protocol from XOR to general symmetric Boolean functions as well as addition over any finite Abelian group, and show that t additions can be performed using only $\tilde{O}(t^2)$ random bits, namely essentially for the same price as one.

Upper bounds for general functions. Finally, building on the techniques with recent randomness-efficient constructions of private circuits [19], we obtain an explicit protocol for evaluating a general circuit C using only $O(t^2 \cdot \log |C|)$ random bits, but in an easier setting that allows for additional "helper parties" who do not contribute any inputs but still participate in the protocol. This upper bound too matches our lower bound up to the logarithmic factor, and gives at least a factor $\Omega(t)$ improvement over previous randomness-efficient MPC protocols from [9,22].

We leave open the question of characterizing the randomness complexity of general MPC without helper parties, as well as closing the remaining (polylogarithmic) gaps between our lower bounds and upper bounds. Evidence for the difficulty of these questions in some parameter regimes was given by Kushilevitz et al. [29], who showed a two-way relation between the randomness complexity of f for $t = 1$ and its circuit complexity. We refer the readers to the full version of this paper [20] for discussion about other related directions.

2 Technical Overview

In this section, we give an overview of the technical ideas behind the main results.

2.1 Background: Secure Multiparty Computation

We consider the standard model of information-theoretic MPC: a set of n parties $\{P_1, P_2, \ldots, P_n\}$, each holding an input x_i from a finite domain D_i, jointly run a protocol Π to compute a function $f : D_1 \times D_2 \times \ldots \times D_n \to Z$. At the end of the protocol, all parties will receive the function output $f(x_1, x_2, \ldots, x_n)$.

During the protocol execution, when needed, each party can toss a random coin and use this random bit in the computation. The randomness complexity of the protocol Π is measured by the total number of random bits that are used during the protocol execution.

In the following, we will use $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ to denote the input, and $\boldsymbol{r} = (r_1, r_2, \ldots, r_n)$ to denote the random tapes of all parties. The function output is denoted by $f(\boldsymbol{x})$, and an execution of the protocol Π with input $\boldsymbol{x}$ and random tapes $\boldsymbol{r}$ is denoted by $\Pi(\boldsymbol{x}, \boldsymbol{r})$.

We consider the standard definition of correctness and semi-honest security.

- The correctness of the protocol Π requires that, when all parties honestly follow the protocol, they will finally output $f(\boldsymbol{x})$ at the end of the protocol.
- Let t be the number of corrupted parties. The semi-honest security of the protocol Π requires that the joint view of any set of t parties can be perfectly simulated by their inputs and the function output.

Note that the semi-honest security implies that, for any set T of t parties, and for all $\boldsymbol{x}, \boldsymbol{x}'$ such that $f(\boldsymbol{x}) = f(\boldsymbol{x}')$ and $x_i = x'_i$ for all $i \in T$, the distribution of the joint view of parties in T of a random execution with input $\boldsymbol{x}$ is identical to that of a random execution with input $\boldsymbol{x}'$.

2.2 Randomness Lower Bound for XOR and Symmetric Functions

To better exhibit our idea, we begin with an n-ary XOR function for simplicity. Concretely, we consider the function $f : (\{0,1\})^n \to \{0,1\}$ defined by

$$f(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n.$$

Suppose Π is an MPC protocol that computes f. Our result shows that any such protocol must use $\Omega(t^2)$ random bits, improving the previous $\Omega(t)$ lower bound from [27] and matching their $O(t^2 \log(n/t))$ upper bound up to at most a logarithmic factor.

We start with the following known fact:

Fact 1. For every P_i, the messages exchanged with P_i together with $f(\boldsymbol{x})$ fully determine its input x_i.

A similar fact was proved and used in [13] to show a lower bound on the communication complexity of the XOR function, and in [6] to show a lower bound on the randomness complexity of the XOR function.[4]

Ideas Behind Fact 1. To see why this fact is true, suppose that there are two executions, $\Pi(\boldsymbol{x}, \boldsymbol{r})$ and $\Pi(\boldsymbol{x}', \boldsymbol{r}')$, such that x_i and x'_i are different, but the messages exchanged with P_i and the function output are identical. Now consider a third execution $\Pi(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$ where $\tilde{\boldsymbol{x}} = \boldsymbol{x}$ except that $\tilde{x}_i = x'_i$, and $\tilde{\boldsymbol{r}} = \boldsymbol{r}$ except that $\tilde{r}_i = r'_i$. I.e., the third execution $\Pi(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$ is the first execution $\Pi(\boldsymbol{x}, \boldsymbol{r})$ except that we replace P_i's input and random tape by those in the second execution $\Pi(\boldsymbol{x}', \boldsymbol{r}')$. Consider the messages exchanged with P_i in these three executions:

- From the point of view of the party P_i, P_i uses the same input and random tape in $\Pi(\boldsymbol{x}', \boldsymbol{r}')$ and $\Pi(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$. Therefore, if P_i always receives the same messages from other parties in these two executions, he cannot distinguish these two executions, and thus will always send the same messages to other parties.
- Similarly, from the point of view of all other parties $\{P_j\}_{j \neq i}$, they use the same input and random tapes in $\Pi(\boldsymbol{x}, \boldsymbol{r})$ and $\Pi(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$. Therefore, if $\{P_j\}_{j \neq i}$ always receive the same messages from P_i in these two executions, they cannot distinguish these two executions, and thus will always send the same messages to P_i.

Note that before the first message exchanged with P_i, P_i cannot distinguish $\Pi(\boldsymbol{x}', \boldsymbol{r}')$ and $\Pi(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$, and all other parties $\{P_j\}_{j \neq i}$ cannot distinguish $\Pi(\boldsymbol{x}, \boldsymbol{r})$ and $\Pi(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$. It implies that the first message exchanged with P_i is always the same in these three executions. Thus, by induction, the messages exchanged with P_i are identical in these three executions.

It follows that parties other than P_i cannot distinguish between $\Pi(\boldsymbol{x}, \boldsymbol{r})$ and $\Pi(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$ at the end of the protocol, which means that they will output the same value in both executions. However, since $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ only differ in the i-th input, for the XOR function f, we must have $f(\boldsymbol{x}) \neq f(\tilde{\boldsymbol{x}})$. It means that at least one of $\Pi(\boldsymbol{x}, \boldsymbol{r})$ and $\Pi(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$ outputs an incorrect result, which contradicts with the correctness of Π. Thus, Fact 1 holds.

With Fact 1, we can view the messages exchanged with P_i together with the function output as an *encoding* of P_i's input x_i. Moreover, we observe that this encoding is t-private, i.e., the distribution of any t messages in a random codeword of x_i is independent of x_i.

Fact 2. For every P_i, the messages exchanged with P_i together with $f(\boldsymbol{x})$ form a t-private encoding of x_i.

[4] [6] focuses on a broader class of functions which they refer to as functions with sensitivity n. The XOR function is a concrete instance in this class.

Ideas Behind Fact 2. Intuitively, it follows from the semi-honest security of $\mathit{\Pi}$: for any t messages, the joint view of the senders and the receivers (other than P_i) of these t messages should not reveal the input of P_i. To formally argue it, we consider the following encoding scheme:

- Let $\boldsymbol{x} = (0, 0, \ldots, 0, 1)$, i.e., all inputs are 0 except the last input is 1. And let $\boldsymbol{x}'$ be the input subject to $x'_i = 1$ and $x'_j = 0$ for all $j \neq i$. Then $f(\boldsymbol{x}) = f(\boldsymbol{x}') = 1$ but $x_i \neq x'_i$.
- The encoding of 0 is the messages exchanged with P_i in a random execution with input $\boldsymbol{x}$. And the encoding of 1 is the messages exchanged with P_i of a random execution with input $\boldsymbol{x}'$.

For $t \leq n-2$ and any t messages, we want to show that the distribution of these t messages in a random codeword of 0 is identical to that in a random codeword of 1. To this end, we consider the set T of t parties which are senders or receivers (other than P_i) of these t messages.

If $P_n \notin T$, then we have $x_j = x'_j = 0$ for all $j \in T$. Since $f(\boldsymbol{x}) = f(\boldsymbol{x}')$, by the semi-honest security of $\mathit{\Pi}$, the distribution of the joint view of parties in T of a random execution with input $\boldsymbol{x}$ is identical to that of a random execution with input $\boldsymbol{x}'$. Note that these t messages are in the joint view of parties in T. Therefore, the distribution of these t messages in a random execution with input $\boldsymbol{x}$ is identical to that in a random execution with input $\boldsymbol{x}'$.

When $P_n \in T$, the above argument fails because $x_n = 1$ while $x'_n = 0$. To fix it, we consider another input $\tilde{\boldsymbol{x}}$ as an intermediate step towards proving the t-privacy. Since $t \leq n-2$, there is a party $P_{i^\star}$ which is not in $T \bigcup \{P_i\}$. We choose $\tilde{\boldsymbol{x}}$ subject to $\tilde{x}_{i^\star} = 1$ and $\tilde{x}_j = 0$ for all $j \neq i^\star$. Then $f(\boldsymbol{x}) = f(\boldsymbol{x}') = f(\tilde{\boldsymbol{x}}) = 1$.

On one hand, since $x_i = \tilde{x}_i = 0$ and $f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}})$, by the semi-honest security of $\mathit{\Pi}$, P_i cannot distinguish a random execution with input $\boldsymbol{x}$ from a random execution with input $\tilde{\boldsymbol{x}}$. Note that these t messages are in the view of P_i. Therefore, the distribution of these t messages in a random execution with input $\boldsymbol{x}$ is identical to that in a random execution with input $\tilde{\boldsymbol{x}}$.

On the other hand, since $x'_j = \tilde{x}_j = 0$ for all $j \in T$ and $f(\boldsymbol{x}') = f(\tilde{\boldsymbol{x}})$, by the semi-honest security of $\mathit{\Pi}$, $\{P_j\}_{j \in T}$ cannot distinguish a random execution with input $\boldsymbol{x}'$ from a random execution with input $\tilde{\boldsymbol{x}}$. Note that these t messages are also in the joint view of parties in T. Therefore, the distribution of these t messages in a random execution with input $\boldsymbol{x}'$ is identical to that in a random execution with input $\tilde{\boldsymbol{x}}$.

Combining these two parts together, we have shown that the above encoding scheme is t-private.

With Fact 2, we are interested in the randomness complexity of a t-private encoding scheme. In our work, we show that for any t-private encoding scheme for a single bit, the support of 0 (i.e., the set of all possible codewords of 0) is of size at least 2^t. In Section 2.2, we will discuss how we prove this result. Jumping ahead, this implies that when the input is $\boldsymbol{x}$, the view of each party P_i has at least 2^t possibilities.

Connection to Randomness Complexity. In [31,21], it has been shown that for a fixed input $\boldsymbol{x}$, if the protocol execution with input $\boldsymbol{x}$ has 2^d different transcripts (i.e., the joint view of all parties), then the protocol uses at least d random bits. Thus, the result that the view of P_i has at least 2^t possibilities implies that the protocol requires at least t random bits.

Final Piece. Indeed, the above result is when we *only* consider the view of a *single* party. We note that, if we fix the view of the first party P_1 (by corrupting P_1), the protocol Π effectively computes the XOR function for the rest of $n-1$ parties that is secure against $t-1$ parties. In particular, we show that the above argument continues to work for the view of the second party: given the view of P_1, the view of P_2 has at least 2^{t-1} possibilities. In general, we show the following:

Fact 3. For all $i \in \{1, 2, \ldots, t\}$, for a fixed input $\boldsymbol{x}$ and given the views of the first $i-1$ parties, the view of P_i has at least 2^{t-i+1} different possibilities.

Thus, for a fixed input $\boldsymbol{x}$, the joint view of the first t parties has at least $\prod_{i=1}^{t} 2^{t-i+1} = 2^{t(t+1)/2}$ different possibilities. It implies that the protocol Π requires $\Omega(t^2)$ random bits.

We note that this lower bound argument holds even if the output is only given to a strict (nonempty) subset of the parties and even if there is an arbitrary number of additional “helper” parties who do not have an input.

Extending to Symmetric Functions. We generalize the previous lower bound to an arbitrary (nontrivial) symmetric Boolean function. For this, it suffices to prove that the above three facts still hold.

- For Fact 1, the main task is to find two executions $\Pi(\boldsymbol{x}, \boldsymbol{r})$ and $\Pi(\boldsymbol{x}', \boldsymbol{r}')$ such that (1) $x_i \neq x_i'$ but the messages exchanged with P_i together with the function output in $\Pi(\boldsymbol{x}, \boldsymbol{r})$ are identical to those in $\Pi(\boldsymbol{x}', \boldsymbol{r}')$, and (2) $f(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n) \neq f(\boldsymbol{x}')$. We show that such two executions exist for any symmetric function that outputs a single bit.
- For Fact 2, it relies on Fact 1 and the semi-honest security of the protocol. Beyond that, we also need to find proper inputs $\boldsymbol{x}, \boldsymbol{x}'$ for the encoding scheme and $\tilde{\boldsymbol{x}}$ that is used to prove the t-privacy of the encoding scheme. We observe that for a symmetric function, we can continue to use the inputs we construct above.
- For Fact 3, it follows from Fact 2 and the randomness complexity of t-private encoding schemes.

Thus, we show that for any non-constant n-ary symmetric function that outputs a single bit, any MPC protocol requires $\Omega(t^2)$ random bits.

Randomness Complexity of t-Private Encoding Schemes Let $(\texttt{Enc}, \texttt{Dec})$ be a t-private encoding scheme for a single bit. Here t-privacy means that the distribution of any t bits in a random codeword is independent of the input bit.

Our goal is to show that the support of 0 (i.e., the set of all possible codewords of 0) is of size at least 2^t. Let $\texttt{supp}(m)$ denote the support of $m \in \{0,1\}$. The lower bound is proved using the following simple inductive argument:

1. When $t = 1$, we show that the support of 0 is of size at least 2. Let c be a codeword of 0 and c' be a codeword of 1. By the correctness of the encoding scheme, $c \neq c'$. Without loss of generality, assume the first bits of c and c' are different. Since the encoding scheme is 1-private, the distribution of the first bit in a random codeword of 0 is identical to that in a random codeword of 1. Then the first bit in a random codeword of 0 is not a constant bit. Otherwise, the first bit in a random codeword of 1 should be the same constant bit, which contradicts with the assumption that the first bits of c and c' are different.
   Since the first bit can take both 0 and 1, there are at least two codewords of 0. The statement holds for $t = 1$.
2. Suppose the statement holds for $t - 1$. With the same argument as above, there exists a bit in a random codeword of 0 which is not a constant bit. Without loss of generality, assume that it is the first bit.
   Since the encoding scheme is t-private, the distribution of any t bits in a random codeword of 0 is identical to that in a random codeword of 1. Then, given the first bit, the encoding scheme is $(t-1)$-private. Thus, according to the induction hypothesis, there are at least 2^{t-1} codewords of 0 given the first bit. Note that the first bit can take both 0 and 1, and in each case, there are at least 2^{t-1} codewords of 0 given the first bit. Thus, there are at least 2^t codewords of 0. The statement holds for t.
3. By induction, we conclude that $|\texttt{supp}(0)| \geq 2^t$.

In the full version of this paper [20], we provide an alternative proof (due to Yuval Filmus) of a slightly weaker lower bound using Fourier analysis. This alternative proof also applies to t-private encodings with imperfect correctness.

2.3 Explicit Randomness Upper Bounds for XOR and Addition

In [27], Kushilevitz and Mansour gave an n-party MPC protocol for the XOR function with semi-honest security against t corrupted parties, which uses $O(t^2 \cdot \log(n/t))$ random bits. This upper bound matches our lower bound, $\Omega(t^2)$ random bits, up to (at most) a logarithmic factor. However, the construction in [27] is non-explicit, relying on a combinatorial object that can either be generated by a probabilistic construction (with small but nonzero failure probability) or generated deterministically in time $(n/t)^{O(t)}$. In this part, we introduce our techniques towards constructing an explicit n-party computation protocol for the XOR function, which uses $O(t^2 \cdot \log^2 n)$ random bits, and where the running time of all parties is polynomial in n.

Basic Protocol. We start with describing the construction in [27]. Following [27], we first assume that there is an ideal functionality $\mathcal{F}_{\text{rand}}$ that generates correlated random bits for all parties. The protocol is as follows:

1. $\mathcal{F}_{\text{rand}}$ first prepare n random bits $r_1, r_2, \ldots, r_n$ subject to $\oplus_{i=1}^{n} r_i = 0$. We will specify the distribution of these n bits later. Then $\mathcal{F}_{\text{rand}}$ sends r_i to P_i.
2. Each party P_i uses r_i to mask its input x_i by computing $g_i = x_i \oplus r_i$. Note that $\oplus_{i=1}^{n} g_i = (\oplus_{i=1}^{n} x_i) \oplus (\oplus_{i=1}^{n} r_i) = \oplus_{i=1}^{n} x_i$. Therefore, the task becomes to compute the XOR of $g_1, g_2, \ldots, g_n$.
3. From $i = 2$ to n, the party P_i receives the partial result $G_{i-1} = \oplus_{j=1}^{i-1} g_j$ from P_{i-1} and computes the partial result $G_i = G_{i-1} \oplus g_i$. Then this result is sent to P_{i+1}. Thus, the last party P_n learns $G_n = \oplus_{i=1}^{n} g_i = \oplus_{i=1}^{n} x_i$ and distributes the function output to all other parties.

The correctness of the protocol follows from the description. As for security, note that when $(r_1, r_2, \ldots, r_n)$ are uniformly random subject to $\oplus_{i=1}^{n} r_i = 0$, $(g_1, g_2, \ldots, g_n)$ are also uniformly random subject to $\oplus_{i=1}^{n} g_i = f(\boldsymbol{x})$, where $f(\boldsymbol{x})$ is the function output. Thus, even learning all $\{g_i\}_{i=1}^{n}$ reveals no information about honest parties' inputs. Therefore, the protocol is secure when $(r_1, r_2, \ldots, r_n)$ are uniformly random subject to $\oplus_{i=1}^{n} r_i = 0$.

Kushilevitz and Mansour [27] noted that, as long as the distribution of the joint view of corrupted parties remains unchanged, we can relax the requirement of the distribution of $\boldsymbol{r} = (r_1, r_2, \ldots, r_n)$ without breaking the security. Concretely, let $\tilde{\boldsymbol{r}} = (\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_n)$ be uniformly random bits subject to $\oplus_{i=1}^{n} \tilde{r}_i = 0$. Let $\texttt{View}(P_i, \boldsymbol{x}, \boldsymbol{r})$ denote the view of P_i in an execution with input $\boldsymbol{x}$ and random bits $\boldsymbol{r}$. A sufficient condition of maintaining the protocol security is that, for all $\boldsymbol{x}$ and for all set T of t parties, the random variables $\boldsymbol{r}$ satisfy that

$$\{\texttt{View}(P_i, \boldsymbol{x}, \boldsymbol{r})\}_{i \in T} \equiv \{\texttt{View}(P_i, \boldsymbol{x}, \tilde{\boldsymbol{r}})\}_{i \in T}.$$

Note that $\texttt{View}(P_i, \boldsymbol{x}, \boldsymbol{r})$ contains (x_i, r_i, G_{i-1}, G_n) (Here G_n is the value received from P_n). Recall that $g_i = x_i \oplus r_i$ for all $i \in \{1, 2, \ldots, n\}$. Given $\boldsymbol{x}$, we are interested in $(r_i, \oplus_{j=1}^{i-1} r_j, \oplus_{j=1}^{n} r_j)$. Let $W = \{r_i, \oplus_{j=1}^{i-1} r_j\}_{i \in T} \bigcup \{\oplus_{j=1}^{n} r_j\}$. Then the above condition can be interpreted as

$$\mathcal{D}(\boldsymbol{r}, W) \equiv \mathcal{D}(\tilde{\boldsymbol{r}}, W),$$

where $\mathcal{D}(\boldsymbol{r}, W)$ and $\mathcal{D}(\tilde{\boldsymbol{r}}, W)$ refer to the distributions of the variables in W instantiated by $\boldsymbol{r}$ and $\tilde{\boldsymbol{r}}$ respectively.[5]

Based on this observation, Kushilevitz and Mansour [27] showed the existence of a sampling space of $\boldsymbol{r}$ of size $(n/t)^{O(t)}$. Therefore, sampling a random $\boldsymbol{r}$ requires $O(t \cdot \log(n/t))$ random bits. Finally, to obtain a protocol in the standard model, it is sufficient to realize $\mathcal{F}_{\text{rand}}$. This is done by letting each of the first $t+1$ parties sample a fresh copy of the random string $\boldsymbol{r}$. Then all parties use the XOR of all random strings in the protocol. Intuitively, since there are at most t corrupted parties, at least one copy of the random string is generated by an honest party, which is unknown to the corrupted parties. Therefore, given the random strings generated by corrupted parties, the XOR of all random strings has the same distribution as that generated by $\mathcal{F}_{\text{rand}}$. In this way, Kushilevitz and Mansour [27] obtained an MPC protocol for XOR with randomness complexity $O(t^2 \cdot \log(n/t))$.

[5] This formalization is from [19].

Parity Sharing Generator [19]. In [19], Goyal et al. generalized the approach of Kushilevitz and Mansour [27] to support any order of computing the XOR of $g_1, g_2, \dots, g_n$.[6] Similarly to [19], our protocol is based on a tree $\mathtt{Tr}$ with n leaf nodes that represents a possible way of computing the parity of n bits. However, unlike [19], for our explicit construction it is crucial that $\mathtt{Tr}$ be a low-depth *full* binary tree (i.e., each node has either two children or no child). Then $\mathtt{Tr}$ has exactly $n-1$ internal nodes and logarithmic depth. The tree $\mathtt{Tr}$ defines the following order of computing the XOR of n bits: All parties start with n bits $g_1, g_2, \dots, g_n$ associated with all leaf nodes. Each time, P_i is responsible to compute the bit associated with the i-th internal node by querying from other parties the bits associated with the two children of the i-th internal node and XORing these two bits. Finally, P_{n-1} computes the bit associated with the root node, which is equal to $\sum_{i=1}^n g_i$.

For a node $v \in \mathtt{Tr}$, let g_v denote the value associated with v, and S_v denote the set of all leaf nodes that are descendants of v. Then $\{g_v\}_{v \in \mathtt{Tr}}$ satisfy that for all internal node v, $g_v = \sum_{i \in S_v} g_i$. For a set T of t corrupted parties, let V be the set of nodes such that for all $v \in V$, g_v is in the joint view of all corrupted parties. Note that the view of each party only contains g_v's for a constant number of nodes v. We have $|V| = O(t)$. Consider the set $W := \{\oplus_{i \in S_v} r_i \mid v \in V\}$. With a similar argument, a sufficient condition of proving security is that, the random variables $\boldsymbol{r}$ satisfy that

$$\mathcal{D}(\boldsymbol{r}, W) \equiv \mathcal{D}(\tilde{\boldsymbol{r}}, W),$$

where $\tilde{\boldsymbol{r}} = (\tilde{r}_1, \tilde{r}_2, \dots, \tilde{r}_n)$ are uniformly random subject to $\oplus_{i=1}^n \tilde{r}_i = 0$.

To generate such random bits $\boldsymbol{r}$, Goyal et al. [19] introduced the notion of *parity sharing generators.*[7]

Definition 1 (Access Set [19]). *An access set $\mathcal{A}$ of a set of random variables $\{r_1, r_2, \dots, r_n\}$ is a set of jointly distributed random variables satisfying the following requirements:*

1. *For all $i \in \{1, 2, \dots, n\}$, $r_i \in \mathcal{A}$.*
2. *Every variable in $\mathcal{A}$ is a linear combination of $r_1, r_2, \dots, r_n$.*

Definition 2 (Parity Sharing Generators [19]). *Let $G : \{0,1\}^m \to \{0,1\}^n$ be a function, $\boldsymbol{u} = (u_1, u_2, \dots, u_m)$ be a vector of random variables in $\{0,1\}^m$ that are uniformly distributed, and $\boldsymbol{r} = (r_1, r_2, \dots, r_n) = G(\boldsymbol{u})$. Let $\mathcal{A}$ be an access set of the random variables $\{r_1, r_2, \dots, r_n\}$. The function G is a t-resilient parity sharing generator with respect to $\mathcal{A}$ if the following holds:*

1. *The output $\boldsymbol{r} = (r_1, r_2, \dots, r_n)$ satisfies that $r_1 \oplus r_2 \oplus \dots \oplus r_n = 0$.*

[6] The work [19] focuses on the private circuits model of [24]. However, it can be transformed to the setting of MPC.

[7] In fact, Goyal et al. [19] introduced the stronger notion of *robust* parity sharing generators, but only gave a probabilistic construction. See more discussion in the full version of this paper [20].

2. *Let $\tilde{\boldsymbol{r}} = (\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_n)$ be a vector of random variables in $\{0,1\}^n$ which are uniformly distributed subject to $\tilde{r}_1 \oplus \tilde{r}_2 \oplus \ldots \oplus \tilde{r}_n = 0$. For any set W of t variables in $\mathcal{A}$, the output $\boldsymbol{r}$ satisfies that*
$$\mathcal{D}(\boldsymbol{r}, W) \equiv \mathcal{D}(\tilde{\boldsymbol{r}}, W),$$
where $\mathcal{D}(\boldsymbol{r}, W)$ and $\mathcal{D}(\tilde{\boldsymbol{r}}, W)$ denote the distributions of the variables in W when they are instantiated by $\boldsymbol{r}$ and $\tilde{\boldsymbol{r}}$ respectively.

Note that when we choose the access set $\mathcal{A} = \{\oplus_{i \in S_v} r_i \mid v \in \texttt{Tr}\}$, the output of an $O(t)$-resilient parity sharing generator with respect to $\mathcal{A}$ satisfies the sufficient condition. Thus, to obtain an explicit MPC protocol for the XOR function, it is sufficient to construct an explicit parity sharing generator with respect to $\mathcal{A}$.

Explicit Construction of Parity Sharing Generators. For a set of random variables $\{r_1, r_2, \ldots, r_n\}$ and a full binary tree $\texttt{Tr}$ with n leaf nodes, an access set $\mathcal{A}$ with respect to $\texttt{Tr}$ is defined by $\mathcal{A} = \{\oplus_{i \in S_v} r_i \mid v \in \texttt{Tr}\}$. We are interested in access sets that are based on full binary trees. Our construction uses a t-wise independent pseudo-random generator in a black box way.

Our idea is to assign a bit to each node in $\texttt{Tr}$ such that for all internal node v and its two children c_0, c_1, the bit assigned to v is equal to the XOR of the bits assigned to c_0 and c_1. Then the bits associated with the leaf nodes are the output. Note that the access set $\mathcal{A}$ consists of the bits associated with all nodes in $\texttt{Tr}$. For a node $v \in \texttt{Tr}$, we use $\mathsf{val}(v)$ to denote the bit associated with v.

Let D be the depth of $\texttt{Tr}$. Our construction works as follows:

1. We start with the root node. We set $\mathsf{val}(\texttt{rt}) = 0$. This ensures that the XOR of the bits associated with all leaf nodes is equal to 0.
2. From $d = 2$ to D, assume that we have assigned bits to nodes of depth $d-1$. Let ℓ_d denote the number of nodes of depth d. Since $\texttt{Tr}$ is a full binary tree, ℓ_d is even. We use $c_1, c_1, \ldots, c_{\ell_d}$ to denote the nodes of depth d such that for all $i \in \{1, 2, \ldots, \ell_d/2\}$, (c_{2i-1}, c_{2i}) are the two children of a node v_i of depth $d-1$.
   Since $\mathsf{val}(c_{2i}) = \mathsf{val}(c_{2i-1}) \oplus \mathsf{val}(v_i)$, we only need to assign a bit to the node c_{2i-1} and then compute the bit associated with c_{2i} accordingly. For $\{c_{2i-1}\}_{i=1}^{\ell_d/2}$, we use the output of a t-wise independent PRG.

Consider a set W of t bits in $\mathcal{A}$. Let $V = \{v \mid \mathsf{val}(v) \in W\}$. Then $|V| = t$. We want to prove that
$$\mathcal{D}(\boldsymbol{r}, \{\mathsf{val}(v)\}_{v \in V}) \equiv \mathcal{D}(\tilde{\boldsymbol{r}}, \{\mathsf{val}(v)\}_{v \in V}).$$

For a node v in $\texttt{Tr}$, we say v is a *left node* if v is a left child of some node in $\texttt{Tr}$. Similarly, we say v is a *right node* if v is a right child of some node in $\texttt{Tr}$. Effectively, we only assign bits to all left nodes in $\texttt{Tr}$. For each depth $d \geq 2$, the bits associated with all left nodes of depth d are t-wise independent. Thus, we want to find a set $V' \subset \texttt{Tr}$ such that V' only contains left nodes and the bits in V' fully determine the bits in V.

Consider the following process:

- For each right node in V, since the bit associated with this node is determined by the bits associated with its left sibling and its parent, we can remove this right node from V and add its left sibling and its parent in V. We repeat the same step for its parent until the parent node is a left node or a root node.
- Note that the bit associated with the root node is a constant 0. We can always remove the root node from V.

In this way, we obtain the set V' that only contains left nodes such that the bits in V' fully determine the bits in V. Thus, it is sufficient to prove that

$$\mathcal{D}(\boldsymbol{r}, \{\mathsf{val}(v)\}_{v \in V'}) \equiv \mathcal{D}(\tilde{\boldsymbol{r}}, \{\mathsf{val}(v)\}_{v \in V'}).$$

We observe that, to remove a right node in V, we may need to insert a left node of each depth. In other words, for all $d \geq 2$, removing a right node in V may insert at most 1 left node of depth d. Therefore, the number of left nodes in V' is bounded by $|V| = t$. Recall that in our construction, we use t-wise independent random bits for all left nodes of each depth. It means that the bits associated with nodes in V' are uniformly random. Thus $\mathcal{D}(\boldsymbol{r}, \{\mathsf{val}(v)\}_{v \in V'})$ is identical to the distribution of $|V'|$ random bits.

We can show that $\mathcal{D}(\tilde{\boldsymbol{r}}, \{\mathsf{val}(v)\}_{v \in V'})$ is also identical to the distribution of $|V'|$ random bits. Intuitively, this is because $\tilde{\boldsymbol{r}}$ is already the most uniform output we can hope. Since $\{\mathsf{val}(v)\}_{v \in V'}$ are uniformly random bits when instantiated by $\boldsymbol{r}$, they should also be uniformly random when instantiated by $\tilde{\boldsymbol{r}}$. Thus, our construction yields a t-resilient parity sharing generator.

Regarding the input size of our construction (i.e., the number of random bits), we need to invoke a t-wise independent PRG for each depth. Therefore, the input size of our construction is D times the input size of a t-wise independent PRG. It is well-known that when the output size is n, there is an explicit t-wise independent PRG with input size $O(t \cdot \log n)$. Also, we can choose to use a full binary tree of depth $\log n$. Therefore, we obtain an explicit construction of a t-resilient parity sharing generator that uses $O(t \cdot \log^2 n)$ random bits. When we use our explicit construction to instantiate the MPC protocol for XOR from [27,19], we obtain an MPC protocol that uses $O(t^2 \cdot \log^2 n)$ random bits.

From a Single Parity to Multiple Additions. All of the above techniques (including the techniques from [27,19] and our techniques of constructing parity sharing generators) can be naturally extended to addition over any Abelian group $\mathbb{G}$, increasing the randomness complexity by a $\log |\mathbb{G}|$ factor. We show that one can in fact do better in the *amortized* setting of computing many additions. Concretely, the asymptotic randomness cost of computing t additions is essentially the same computing a single addition. We outline the techniques below.

First, we naturally extend the notion of a parity sharing generator to a general Abelian group $\mathbb{G}$, referring to the generalized notion as a *zero sharing generator*. We show that our technique also yields an explicit construction of zero-sharing generator. We then amortize the randomness complexity by using the following natural randomness extraction approach. Consider the case of $\mathbb{Z}_2$ for simplicity. Suppose all parties want to compute the XOR function ℓ times. We can first prepare the random strings $\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}, \ldots, \boldsymbol{r}^{(\ell)}$ in a batch way, and then use one fresh copy in each execution. Finally, we use a t-resilient randomness extractor $\texttt{Ext} : \{0,1\}^m \to \{0,1\}^\ell$ for bit-fixing sources [12], guaranteeing that when the input is randomly sampled from $\{0,1\}^m$, the output is uniformly random even when conditioned on any t input bits.

To prepare the random strings $\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}, \ldots, \boldsymbol{r}^{(\ell)}$, we will let each P_i of the first m parties distribute a fresh copy of the random string, denoted by $\boldsymbol{\tau}^{(i)}$. Then all parties use $\texttt{Ext}$ to extract ℓ random strings. By the property of a t-resilient randomness extractor, the output strings $\{\boldsymbol{r}^{(i)}\}_{i=1}^{\ell}$ are random given the random strings $\{\boldsymbol{\tau}^{(i)}\}_{i \in T}$ generated by corrupted parties.

It is known that there is a t-resilient randomness extractor based on Vandermonde matrices with input size $m = \ell + t \cdot \log(\ell + t)$. Thus, we obtain an MPC protocol for ℓ XOR computations that uses $O((\ell + t \cdot \log(\ell + t)) \cdot t \cdot \log^2 n)$ random bits, giving an amortized cost of only $O(t \cdot \log^2 n)$ random bits per XOR.

2.4 Upper Bounds Beyond Linear Functions

The previous upper bounds apply only to linear functions over an Abelian group. Building on these results, we obtain near-optimal upper bounds for general symmetric functions, or even general circuits if additional "helper parties" are allowed.

Upper Bound for Symmetric Functions. For any symmetric function $f : \{0,1\}^n \to \{0,1\}$, we show that there is an explicit MPC protocol that uses $O(t^2 \cdot \log^3 n)$ random bits. This includes useful functions such as majority or threshold, and matches the previous lower bound for nontrivial symmetric functions up to a polylogarithmic term. Our protocol uses the standard Shamir secret sharing scheme and the BGW protocol [4,11]. We will use $[r]_t$ to denote a degree-t Shamir sharing of r. Our idea works for all $t < \frac{n}{\lceil \log n \rceil}$:

1. For a symmetric function f, the output only depends on the number of 1s in the input bits. Let p be a prime such that $n < p < 2n$. Consider the finite field $\mathbb{F}_p$. All parties will first compute a degree-t Shamir secret sharing of the summation of all input bits in $\mathbb{F}_p$, denoted by $[s]_t$. This is achieved by the following steps:
   (a) All parties first prepare a random degree-t Shamir sharing $[r]_t$ by letting each of the first $t+1$ parties distributes a random degree-t Shamir sharing and using the summation of these $t+1$ sharings. They transform $[r]_t$ to a random additive sharing by locally multiplying proper Lagrange coefficients with their shares.
   (b) All parties compute the summation of all input bits together with all shares of the random additive sharing by using our protocol for addition over $\mathbb{F}_p$ (recall that we extend the protocol for XOR to addition over any Abelian group). Then the output is equal to $s + r$, where s is the summation of all input bits.
   (c) Finally, all parties locally compute $[s]_t = (s + r) - [r]_t$.

2. Note that s is the number of 1s in the input bits, and $s \in \{0, 1, \ldots, n\}$. Therefore, there exists a function $g : \{0, 1, \ldots, n\} \to \{0, 1\}$ such that $f(\boldsymbol{x}) = g(s)$, where $s = \sum_{i=1}^{n} x_i$. We note that g can be represented by a degree-n polynomial in $\mathbb{F}_p$. Our idea is to compute a Shamir sharing of the output $g(s)$.
   (a) All parties first use the BGW protocol to compute $[s^{2^i}]_t$ for all $i \in \{0, 1, \ldots, \lceil \log n \rceil - 1\}$. This step requires $O(\log n)$ multiplications.
   (b) Then, all parties can use $\{[s^{2^i}]_t\}_{i=0}^{\lceil \log n \rceil - 1}$ to locally compute a Shamir sharing of s^j for all $j \in \{1, 2, \ldots, n\}$. In particular, the resulting sharing has degree at most $t \cdot \lceil \log n \rceil < n$. Therefore, the resulting sharing can still be reconstructed by all parties. Thus, they can locally compute a Shamir sharing of the output $g(s)$ of degree at most $t \cdot \lceil \log n \rceil < n$.
3. Finally, all parties reconstruct the Shamir sharing of $g(s)$. This is achieved by first transforming it to an additive sharing of $g(s)$ and then using our protocol for addition over $\mathbb{F}_p$.

In summary, we need 2 invocations of the addition protocol over $\mathbb{F}_p$ and $O(\log n)$ multiplications using the BGW protocol [4] (the preparation of a degree-t Shamir sharing costs the same amount of randomness as doing 1 multiplication in [4]). In [4], doing $O(\log n)$ multiplications require $O(t^2 \cdot \log n)$ random field elements. Our addition protocol over $\mathbb{F}_p$ requires $O(t^2 \cdot \log^2 n)$ random field elements. Since each element in $\mathbb{F}_p$ is of size $O(\log n)$, for any symmetric function, we obtain an explicit construct that uses $O(t^2 \cdot \log^3 n)$ random bits. We refer the readers to the full version of this paper [20] for more details.

Upper Bound for General Circuits with Helper Parties. Finally, we consider the goal of evaluating general functions in a relaxed setting where there are extra helper parties that can participate in the protocol but do not have inputs nor receive the output. In this model, we give an explicit MPC protocol for a general circuit C that uses $O(t^2 \cdot \log |C|)$ random bits, where $|C|$ is the circuit size. Since our lower bound for XOR extends to the setting of helper parties, this upper bound is essentially optimal.

Our construction uses a variant of the private circuits model from [24] referred to as a *leakage-tolerant* private circuit [22,1], building on the recent randomness-efficient construction from [19].[8] Informally, a leakage-tolerant private circuit with (unprotected) input x and output y is a randomized circuit such that the values of any t internal wire values can be simulated by probing t input and output wires. Letting each party simulate a single gate in such a tolerant circuit, we obtain an MPC protocol with helper parties in which corrupting t parties reveals at most t inputs and outputs. Note that it does *not* directly give us an

[8] In the current context, one could plausibly use the explicit construction of a private circuit with quadratic randomness complexity in [14] as a substitute for the quasilinar-randomness construction from [19]. However, the analysis of [14] only considers standard leakage-resilience whereas here we need the stronger leakage-tolerance property analyzed in [19].

MPC protocol in the usual sense, since the revealed inputs and outputs may belong to honest parties.

Our idea is to first let all parties secret-share their inputs among the helper parties. Then all helper parties together emulate a leakage-tolerant private circuit to compute a secret-sharing of the function output. Finally, the output is reconstructed to the parties who should receive it. To make this idea work, we need to design an efficient protocol that allows parties to secret-share their inputs:

1. We note that, for each party P_i, it is sufficient to use a t-private encoding of its input. This is because corrupting any t helper parties reveals at most t input and output values, which are independent of P_i's input. We borrow the encoding scheme from [19], which is based on a strong t-wise independent PRG. It requires $O(t \cdot \log m)$ random bits to encode m bits.
2. However, we cannot afford the cost of allowing each party to use fresh random seeds to encode their inputs, since it requires $O(t \cdot n \cdot \log m)$ random bits. We observe that all parties can actually use t-wise independent random seeds. This is because each corrupted party who holds an input only observes its own random seed, and each corrupted helper party receives at most one bit of the encoding of some input. Thus, the joint view of corrupted parties depends on the encoding of at most t inputs, which in turn depend on at most t random seeds. Therefore, t-wise independent random seeds are sufficient. Generating these random seeds (via a trusted party) require $O(t^2 \cdot \log m)$ random bits.
3. Finally, note that we *cannot* use the same method as that in [27] to generate these random seeds in a distributed way because the random seeds have size $O(t^2 \cdot \log m)$. If we ask each of the first $t+1$ parties to generate a fresh copy of the random seeds, we would need $O(t^3 \cdot \log m)$ random bits. Our idea is to use a t-resilient randomness extractor. We ask each of the first $2t$ parties to generate t-wise independent random seeds of size $O(t \cdot \log m)$. Then, all parties use a t-resilient randomness extractor to extract t copies of fresh random seeds. Finally, each party concatenates its t copies and obtains a random seed of length $O(t^2 \cdot \log m)$.

We use the construction of a leakage-tolerant private circuit from [19], which uses $O(t \cdot \log t|C|)$ random bits. Since the input size m is upper bounded by the circuit size, we obtain an MPC protocol for a general circuit that uses only $O(t^2 \cdot \log |C|)$ random bits.

As a final challenge, note that the leakage-tolerant private circuit in [19] is not explicit. In the full version of this paper [20], we show that our technique allows us to obtain an explicit multi-phase parity sharing generator, which outputs multiple additive sharings of 0. Then, we show how to use our explicit construction of multi-phase parity sharing generators to instantiate the private circuit in [19]. The instantiation only requires $O(t^2 \cdot \log^2 t|C|)$ random bits. We use it to obtain an *explicit* construction of an MPC protocol (with helper parties) for a general circuit C that uses $O(t^2 \cdot \log^2 t|C|)$ random bits.

3 Preliminaries

3.1 Secure Multiparty Computation

In this work, we consider the setting where a set of n parties, $\{P_1, P_2, \ldots, P_n\}$, each holding an input x_i from a finite domain D_i, jointly run a protocol to compute a function $f : D_1 \times \ldots \times D_n \to Z$. At the end of the protocol, all parties receive the function output $f(x_1, \ldots, x_n)$.

Each party has a private random tape which contains uniformly random bits. We use $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ to denote the inputs of all parties and $\boldsymbol{r} = (r_1, r_2, \ldots, r_n)$ to denote the random tapes of all parties. For a party P_i, we use $\texttt{View}(P_i, \boldsymbol{x}, \boldsymbol{r})$ to denote the information that is observed by P_i in an execution with inputs $\boldsymbol{x}$ and random tapes $\boldsymbol{r}$, which includes his input, random tape, messages received from other parties, and the function output. We use $\texttt{View}(P_i, \boldsymbol{x})$ to denote the random variable over the distribution induced by $\texttt{View}(P_i, \boldsymbol{x}, \boldsymbol{r})$ when $\boldsymbol{r}$ is sampled uniformly.

In this work, we consider perfect correctness and semi-honest security with perfect privacy, defined as follows.

Definition 3 (Correctness and Security). *Let $f : D_1 \times \ldots \times D_n \to Z$ be an n-ary function. For an n-party computation protocol Π that computes f,*

- *(Correctness). We say Π achieves perfect correctness if for all input $\boldsymbol{x}$, when all parties honestly follow the protocol Π, they will finally output $f(\boldsymbol{x})$.*
- *(Security). We say Π achieves semi-honest security with perfect privacy if for all set T of at most t parties, and for all input $\boldsymbol{x}$, there is a probabilistic algorithm $\mathcal{S}$, which takes as input the inputs of parties in T and the function output, and outputs the views of parties in T, such that the following two distributions are identical:*

$$\{\mathcal{S}(\{x_i\}_{i\in T}, f(\boldsymbol{x})), f(\boldsymbol{x})\} \equiv \{\{\texttt{View}(P_i, \boldsymbol{x})\}_{i\in T}, f(\boldsymbol{x})\}.$$

If Π achieves both perfect correctness and semi-honest security with perfect privacy, we say Π achieves perfect semi-honest security.

Intuitively, the security requires that the joint view of all corrupted parties only depends on their inputs and the function output. We have the following property of a protocol Π with semi-honest security and perfect privacy.

Property 1. Let $f : D_1 \times \ldots D_n \to Z$ be an n-ary function. Let Π be an n-party protocol that computes f with semi-honest security and perfect privacy against t corrupted parties. Then for all set T of at most t parties, and for all $\boldsymbol{x}, \boldsymbol{x}' \in D_1 \times \ldots \times D_n$ such that $f(\boldsymbol{x}) = f(\boldsymbol{x}')$ and $x_i = x'_i$ for all $i \in T$, the following two distributions are identical:

$$\{\texttt{View}(P_i, \boldsymbol{x})\}_{i\in T} \equiv \{\texttt{View}(P_i, \boldsymbol{x}')\}_{i\in T}.$$

Randomness Complexity of a Protocol. We follow the definition of randomness complexity from [27]. At the beginning of the protocol, each party has a private random tape that contains uniformly random bits. Each time a party needs to use a random bit, he reads the rightmost unused bit on his random tape. Note that each party may use different number of random bits in different executions. The number of random bits that is used by the protocol is the total number of random bits used by all parties. The randomness complexity is the worst case (over all inputs and all executions) number of random bits. The same model for randomness complexity is also used in [29,31,21].

We will use the following lemma from [31,21].

Lemma 1 ([31,21]). *For a given input $\boldsymbol{x}$, let d be the maximum, over all protocol executions on $\boldsymbol{x}$, of the number of random bits used by all parties during a given execution. Then, the number of different transcripts (i.e., the joint view of all parties) of the protocol execution on $\boldsymbol{x}$ is at most 2^d.*

For some of our positive results, it is convenient to use a natural generalization of this model where parties can sample a uniform value from $\{1, 2, \ldots, p\}$ for any choice of integer $p > 1$. We assume that $\lceil p \rceil = O(\log p)$ random bits are consumed. This can be justified by either entropy considerations, or by the fact that $O(\log p)$ random bits are sufficient to generate a uniform value from $\{1, 2, \ldots, p\}$ in expectation [26,9].

We note that our lower bound also applies to the generalized model with the help of Lemma 1 in the generalized model, of which we provide a proof in the full version of this paper [20].

Helper Parties. We also consider a general model where there are extra k parties $\{P_{n+1}, P_{n+2}, \ldots, P_{n+k}\}$. These parties can participate in the computation but do not have inputs, nor receive the output. We refer to these parties as *helper parties*. The randomness complexity of a protocol in the general model also counts the random bits used by helper parties. The perfect semi-honest security in the general model is defined similarly.

Functions with Minimal Input Domain. For a party P_i, and two distinct inputs $x_i \neq x'_i$, we say a function f is sensitive to (P_i, x_i, x'_i) if there exists $\{x_j\}_{j \neq i}$ such that

$$f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) \neq f(x_1, \ldots, x_{i-1}, x'_i, x_{i+1}, \ldots, x_n).$$

We say a function f has minimal input domain if f is sensitive to all possible (P_i, x_i, x'_i).

Note that if f is not sensitive to (P_i, x_i, x'_i), it means that the function behaves identically on input x_i and x'_i. Then, P_i can always use x_i when his input is x'_i without changing the output of the function, which reduces the size of P_i's input domain. Thus, for a function f that is *not* sensitive to all (P_i, x_i, x'_i), we can repeat the above step and reduce the input domain of f. Therefore, without loss of generality, it is sufficient to only consider functions with minimal input domain.

Symmetric Functions. We say a function f is a symmetric function if it satisfies that:

- All inputs have the same input domain. I.e., $D_1 = D_2 = \ldots = D_n$.
- The output of the function f is independent of the order of the inputs. I.e., for all $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\boldsymbol{x}' = (x'_1, x'_2, \ldots, x'_n)$, where $\boldsymbol{x}'$ is a permutation of $\boldsymbol{x}$, $f(\boldsymbol{x}) = f(\boldsymbol{x}')$.

3.2 t-Private Encoding Schemes

Definition 4 (Encoding Scheme). *Let ℓ, n be positive integers. Let $\mathcal{M} \subset \{0,1\}^\ell$ be the message space and $\mathcal{C} \subset \{0,1\}^n$ be the codeword space. An* encoding scheme *consists of a pair of algorithms* $(\mathtt{Enc}, \mathtt{Dec})$ *where:*

- $\mathtt{Enc}$ *is a randomized algorithm which takes as input a message $m \in \mathcal{M}$ and a random tape $r \in \mathcal{R}$, and outputs a codeword $c \in \mathcal{C}$, denoted by $c = \mathtt{Enc}(m; r)$. When r is not important in the context, we will omit r and simply write $c = \mathtt{Enc}(m)$.*
- $\mathtt{Dec}$ *is a deterministic algorithm which takes as input a codeword $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$.*

The correctness of an encoding scheme requires that for all $m \in \mathcal{M}$, the following holds:

$$\Pr[\mathtt{Dec}(\mathtt{Enc}(m)) = m] = 1$$

Definition 5 (t-Private Encoding Scheme). *We say an encoding scheme* $(\mathtt{Enc}, \mathtt{Dec})$ *is t-private, if for all $m, m' \in M$ and for all t indices $i_1, i_2, ..., i_t \in \{1, 2, \ldots, n\}$, the following two distributions are identical:*

$$\{c \leftarrow \mathtt{Enc}(m) : c[i_1], c[i_2], \ldots, c[i_t]\} \equiv \{c' \leftarrow \mathtt{Enc}(m') : c'[i_1], c'[i_2], \ldots, c'[i_t]\},$$

where $c[i]$ (resp., $c'[i]$) is the i-th bit of c (resp., c').

Strong t-wise Independent Pseudo-random Generators. Our work will use the standard notion of (strong) t-wise independent pseudo-random generators.

Definition 6 ((Strong) t-wise Independent PRG). *Let $\mathbb{G}$ be a finite Abelian group. A function $G : \mathbb{G}^\ell \rightarrow \mathbb{G}^n$ is a t-wise independent pseudo-random generator (or t-wise independent PRG for short) if any subset of t group elements of $G(x)$ are uniformly random and independently distributed when x is uniformly sampled from $\mathbb{G}^\ell$.*

If any subset of t group elements of $(x, G(x))$ are uniformly random and independently distributed when x is uniformly sampled from $\mathbb{G}^\ell$, then we say G is a strong t-wise independent PRG.

We say that a (strong) t-wise independent PRG G is linear *if every output group element is a linear combination of the input group elements. In particular, a linear (strong) t-wise independent PRG G satisfies that for all $x, x' \in \mathbb{G}^\ell$, $G(x) + G(x') = G(x + x')$.*

For a finite field $\mathbb{F}$, it is well known that there is a linear and strong t-wise independent PRG $G : \mathbb{F}^\ell \to \mathbb{F}^n$ based on Reed-Solomon codes with input size $\ell = O(t \cdot \log n)$. (See [19] for a construction over binary field, which can be extended to any finite field.)

Theorem 1. *Let $\mathbb{F}$ be a finite field and n, t be positive integers. Then there is a linear and strong t-wise independent PRG $G : \mathbb{F}^\ell \to \mathbb{F}^n$ with input size $\ell = O(t \cdot \log n)$.*

Randomness Efficient t-Private Encoding Scheme. We borrow the following linear t-private encoding scheme from [19].

Let $G : \{0,1\}^\ell \to \{0,1\}^n$ be a linear and strong t-wise independent PRG. The encoding scheme $(\mathtt{Enc}, \mathtt{Dec})$ works as follows:

- The message space is $\mathcal{M} = \{0,1\}^n$ and the codeword space is $\mathcal{C} = \{0,1\}^{\ell+n}$.
- The encoder Enc takes $\boldsymbol{x} \in \{0,1\}^n$ as input and $\boldsymbol{\rho} \in \{0,1\}^\ell$ as random tape. Then
$$\mathtt{Enc}(\boldsymbol{x}; \boldsymbol{\rho}) = (\boldsymbol{\rho}, G(\boldsymbol{\rho}) \oplus \boldsymbol{x}).$$
- The decoder Dec takes $(\boldsymbol{c}_1, \boldsymbol{c}_2) \in \{0,1\}^\ell \times \{0,1\}^n$ as input and outputs
$$\mathtt{Dec}(\boldsymbol{c}_1, \boldsymbol{c}_2) = G(\boldsymbol{c}_1) \oplus \boldsymbol{c}_2.$$

The linearity follows from that the t-wise independent PRG G is linear. As for t-privacy, since G is a strong t-wise independent PRG, any t bits of $(\boldsymbol{\rho}, G(\boldsymbol{\rho}))$ are uniformly random when $\boldsymbol{\rho}$ is uniformly sampled from $\{0,1\}^\ell$. Therefore, any t bits of $(\boldsymbol{\rho}, G(\boldsymbol{\rho}) \oplus \boldsymbol{x})$ are also uniformly random and thus, independent of $\boldsymbol{x}$.

3.3 Zero Sharing Generators

We first define the notion of access set of a set of random variables $\{r_1, r_2, \ldots, r_n\}$.

Definition 1 (Access Set [19]). *An access set $\mathcal{A}$ of a set of random variables $\{r_1, r_2, \ldots, r_n\}$ is a set of jointly distributed random variables satisfying the following requirements:*

1. *For all $i \in \{1, 2, \ldots, n\}$, $r_i \in \mathcal{A}$.*
2. *Every variable in $\mathcal{A}$ is a linear combination of $r_1, r_2, \ldots, r_n$.*

Let $\boldsymbol{r} = (r_1, r_2, \ldots, r_n)$. For a set $W \subset \mathcal{A}$, we use $\mathcal{D}(\boldsymbol{r}, W)$ to denote the distribution of the variables in W when they are instantiated by $\boldsymbol{r}$.

We follow [19] and define the notion of zero sharing generators. In [19], Goyal, et al focuses on the binary field. We extend this notion to any finite Abelian group $\mathbb{G}$.

Definition 7 (Zero Sharing Generators [19]). *Let $\mathbb{G}$ be a finite Abelian group. Let $G : \mathbb{G}^m \to \mathbb{G}^n$ be a function, $\boldsymbol{u} = (u_1, u_2, \ldots, u_m)$ be a vector of random variables in $\mathbb{G}^m$ that are uniformly distributed, and $\boldsymbol{r} = (r_1, r_2, \ldots, r_n) = G(\boldsymbol{u})$. Let $\mathcal{A}$ be an access set of the random variables $\{r_1, r_2, \ldots, r_n\}$. The function G is a t-resilient zero sharing generator with respect to $\mathcal{A}$ if the following holds:*

1. *The output* $\boldsymbol{r} = (r_1, r_2, \ldots, r_n)$ *satisfies that* $r_1 + r_2 + \ldots + r_n = 0$.
2. *Let* $\tilde{\boldsymbol{r}} = (\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_n)$ *be a vector of random variables in* $\mathbb{G}^n$ *which are uniformly distributed subject to* $\tilde{r}_1 + \tilde{r}_2 + \ldots + \tilde{r}_n = 0$. *For any set* W *of* t *variables in* $\mathcal{A}$, *the output* $\boldsymbol{r}$ *satisfies that*
$$\mathcal{D}(\boldsymbol{r}, W) \equiv \mathcal{D}(\tilde{\boldsymbol{r}}, W),$$
where $\mathcal{D}(\boldsymbol{r}, W)$ *and* $\mathcal{D}(\tilde{\boldsymbol{r}}, W)$ *denote the distributions of the variables in* W *when they are instantiated by* $\boldsymbol{r}$ *and* $\tilde{\boldsymbol{r}}$ *respectively.*

One can view a t-resilient zero sharing generator as a generalization of a t-wise independent PRG in the following two ways:

- First, the output vector should satisfies that the summation of all entries is equal to 0.
- Second, for a t-wise independent PRG, one may think that there is an adversary which can access any t entries in the output vector. A t-resilient zero sharing generator allows an adversary to access any t variables in the access set $\mathcal{A}$ which contains all entries of the output vector.

We can extend a t-resilient zero sharing generator to generating multiple zero sharings with different number of shares as follows.

Definition 8 (Multi-Phase Zero Sharing Generators [19]). *Let* $\mathbb{G}$ *be a finite Abelian group. Let* p *and* $n_1, n_2, \ldots, n_p$ *be positive integers,* $G : \mathbb{G}^m \to \mathbb{G}^{n_1} \times \mathbb{G}^{n_2} \times \ldots \times \mathbb{G}^{n_p}$ *be a function,* $\boldsymbol{u} = (u_1, u_2, \ldots, u_m)$ *be a vector of random variables in* $\mathbb{G}^m$ *that are uniformly distributed, and* $\boldsymbol{r} = (\boldsymbol{r}^{(1)}, \ldots, \boldsymbol{r}^{(p)}) = G(\boldsymbol{u})$ *where* $\boldsymbol{r}^{(j)} = (r_1^{(j)}, \ldots, r_{n_j}^{(j)})$ *for all* $j \in \{1, 2, \ldots, p\}$. *For each* $\boldsymbol{r}^{(j)}$, *let* $\mathcal{A}_j$ *be an access set of the random variables* $\{r_1^{(j)}, \ldots, r_{n_j}^{(j)}\}$, *and* $\mathcal{A} = \bigcup_{j=1}^{p} \mathcal{A}_j$. *The function* G *is a multi-phase* t*-resilient zero sharing generator with respect to* $\mathcal{A}$ *if the following holds:*

1. *For all* $j = \{1, 2, \ldots, p\}$, *the output vector* $\boldsymbol{r}^{(j)} = (r_1^{(j)}, \ldots, r_{n_j}^{(j)})$ *satisfies* $r_1^{(j)} + \ldots + r_{n_j}^{(j)} = 0$.
2. *Let* $\tilde{\boldsymbol{r}} = (\tilde{\boldsymbol{r}}^{(1)}, \ldots, \tilde{\boldsymbol{r}}^{(p)}) \in \mathbb{G}^{n_1} \times \ldots \times \mathbb{G}^{n_p}$ *be uniformly random variables such that for all* $j = \{1, 2, \ldots, p\}$, *the vector* $\tilde{\boldsymbol{r}}^{(j)} = (\tilde{r}_1^{(j)}, \ldots, \tilde{r}_{n_j}^{(j)})$ *satisfies* $\tilde{r}_1^{(j)} + \ldots + \tilde{r}_{n_j}^{(j)} = 0$. *For any set* W *of* t *variables in* $\mathcal{A}$, *the output* $\boldsymbol{r}$ *satisfies*
$$\mathcal{D}(\boldsymbol{r}, W) \equiv \mathcal{D}(\tilde{\boldsymbol{r}}, W),$$
where $\mathcal{D}(\boldsymbol{r}, W)$ *and* $\mathcal{D}(\tilde{\boldsymbol{r}}, W)$ *denote the distributions of the variables in* W *when instantiated by* $\boldsymbol{r}$ *and* $\tilde{\boldsymbol{r}}$ *respectively.*

We say a (multi-phase) t-resilient zero sharing generator G is linear if every output group element is a linear combination of the input group elements. In particular, a linear (multi-phase) t-resilient zero sharing generator G satisfies that for all $\boldsymbol{u}, \boldsymbol{u}' \in \mathbb{G}^m$, $G(\boldsymbol{u}) + G(\boldsymbol{u}') = G(\boldsymbol{u} + \boldsymbol{u}')$.

Tree Based Access Sets. In our work, we are interested in access sets that are based on full binary trees. A full binary tree $\mathtt{Tr}$ satisfies that every node has either no children (i.e., a leaf node) or 2 children. For a set of random variables $\{r_1, r_2, \ldots, r_n\}$ and a full binary tree $\mathtt{Tr}$ with n leaf nodes, an access set $\mathcal{A}$ with respect to $\mathtt{Tr}$ is defined as follows: We first associate the i-th leaf node with the random variable r_i. Then, each internal node is associated with a random variable which is equal to the sum of the random variables associated with its two children. The set $\mathcal{A}$ contains the random variables associated with all nodes in $\mathtt{Tr}$.

4 Lower Bound for Symmetric Functions

In this section we prove our main lower bound, improving over the previous lower bound of [27]. We start with a technical lemma about the randomness complexity of a t-private encoding scheme and then use it to obtain the lower bound.

4.1 Lower bound for t-private Encoding Schemes

In this section, we discuss the randomness complexity of a t-private encoding scheme. We focus on t-private encoding schemes that encode a single bit. We will show that, for any t-private encoding scheme and any input bit $m \in \{0, 1\}$, the number of codewords of m is at least 2^t. Note that it implies that any such a t-private encoding scheme requires at least t random bits. This result will be used to prove the lower bound of the randomness complexity of secure multiparty computation in the next section.

Lemma 2. *For any t-private encoding scheme* $(\mathtt{Enc}, \mathtt{Dec})$ *and any bit* $m \in \{0, 1\}$*,* $|\mathtt{supp}(m)| \geq 2^t$*.*

Proof. We prove the lemma by induction.

When $t = 1$, we show that the support of 0 is of size at least 2. Let c be a codeword of 0 and c' be a codeword of 1. By the correctness of the encoding scheme, $c \neq c'$. Without loss of generality, assume the first bits of c and c' are different. Since the encoding scheme is 1-private, the distribution of the first bit in a random codeword of 0 is identical to that in a random codeword of 1. Then the first bit in a random codeword of 0 is not a constant bit. Otherwise, the first bit in a random codeword of 1 should be the same constant bit, which contradicts with the assumption that the first bits of c and c' are different. Since the first bit can take both 0 and 1, there are at least two codewords of 0. The statement holds for $t = 1$.

Now suppose the statement holds for $t-1$, i.e., for any $(t-1)$-private encoding scheme $(\mathtt{Enc}, \mathtt{Dec})$ and any bit $m \in \{0, 1\}$, $|\mathtt{supp}(m)| \geq 2^{t-1}$. Consider a t-private encoding scheme $(\mathtt{Enc}, \mathtt{Dec})$. With the same argument as above, there exists a bit in a random codeword of 0 which is not a constant bit. Without loss of generality, assume that it is the first bit.

Consider the following encoding scheme $(\texttt{Enc}', \texttt{Dec}')$:

- For $m \in \{0,1\}$, $\texttt{Enc}'(m)$ outputs a random codeword $c = \texttt{Enc}(m)$ subject to $c[1] = 0$. Here $c[1]$ refers to the first bit of the codeword c.
- $\texttt{Dec}' = \texttt{Dec}$.

We show that $(\texttt{Enc}', \texttt{Dec}')$ is a $(t-1)$-private encoding scheme. Let $\texttt{supp}'(m)$ denote the set of codewords of m defined by $(\texttt{Enc}', \texttt{Dec}')$.

The correctness of $(\texttt{Enc}', \texttt{Dec}')$ follows from the correctness of $(\texttt{Enc}, \texttt{Dec})$: if there exists a codeword $c \in \texttt{supp}'(m)$ such that $\texttt{Dec}'(c) \neq m$, since $\texttt{supp}'(m)$ is a subset of $\texttt{supp}(m)$ and $\texttt{Dec}' = \texttt{Dec}$, we have $c \in \texttt{supp}(m)$ and $\texttt{Dec}(c) \neq m$, which contradicts with the correctness of $(\texttt{Enc}, \texttt{Dec})$.

As for $(t-1)$-privacy, recall that $(\texttt{Enc}, \texttt{Dec})$ is t-private. Therefore, for any t bits, the distribution of these t bits in $c = \texttt{Enc}(0)$ is identical to the distribution of these t bits in $c' = \texttt{Enc}(1)$. Then, fixing the first bit to be 0, for any $t-1$ bits, the distribution of these $t-1$ bits in $c = \texttt{Enc}(0)$ subject to $c[1] = 0$ is identical to the distribution of these $t-1$ bits in $c' = \texttt{Enc}(1)$ subject to $c'[1] = 0$. Recall that $\texttt{Enc}'(m)$ outputs a random codeword $c = \texttt{Enc}(m)$ subject to $c[1] = 0$. Therefore $(\texttt{Enc}', \texttt{Dec}')$ is $(t-1)$-private.

According to the induction hypothesis, $|\texttt{supp}'(0)| \geq 2^{t-1}$. I.e., there are 2^{t-1} different codewords in $\texttt{supp}(0)$ whose first bit is 0. By the same argument, there are 2^{t-1} different codewords in $\texttt{supp}(0)$ whose first bit is 1. Therefore, $|\texttt{supp}(0)| \geq 2^t$.

By induction, we conclude that the lemma holds for all t.

We note the following direct corollary.

Corollary 1. *Any t-private encoding scheme* $(\texttt{Enc}, \texttt{Dec})$ *uses at least t random bits.*

Proof. According to Lemma 2, $|\texttt{supp}(0)| \geq 2^t$. Therefore, $\texttt{Enc}(0)$ has at least 2^t different output. Thus the random seed has length at least t.

In the full version of this paper [20], we give an alternative proof (due to Yuval Filmus) of a variant of Lemma 2 by relying on Fourier analysis of Boolean functions and a known bound on the number of roots of a low-degree polynomial over the Boolean hypercube. This variant applies also to t-private encoding with imperfect correctness, to which the above simple combinatorial argument does not apply.

4.2 Randomness Lower Bound for Symmetric Functions

In this section we prove a lower bound on the randomness complexity of secure multiparty computation protocols that compute symmetric functions with a single output bit. This includes parity and threshold functions (including AND, OR, majority) as special cases.

Theorem 2. *For all $n \geq 3$ and $t \leq n-2$, and for all non-constant symmetric functions f that outputs a single bit, any n-party protocol Π that computes f with perfect semi-honest security against t corrupted parties requires at least $\frac{t^2}{2}$ random bits. Moreover, this holds even with an arbitrary number k of helper parties.*

Proof. Recall that, without loss of generality, it is sufficient to only consider functions with minimal input domain. In the following, we assume that f is a non-constant symmetric function with minimal input domain. Without loss of generality, we assume that in every round, each party sends a message in $\{0, 1, \perp\}$ to every other party. This can be achieved by requiring that in each round, every party P_i sends a $\perp$ to every party P_j if P_i does not need to send any bit to P_j in this round, which does not change the randomness complexity of the protocol.

Note that an execution is determined by the inputs and random tapes of all parties. For an execution with inputs $\boldsymbol{x}$ and random tapes $\boldsymbol{r}$, we use $M_{P_i}(\boldsymbol{x}, \boldsymbol{r})$ to denote the messages that P_i receives from or sends to other parties. We use $M_{P_i}(\boldsymbol{x})$ to denote the random variable over the distribution induced by $M_{P_i}(\boldsymbol{x}, \boldsymbol{r})$ when $\boldsymbol{r}$ is sampled uniformly.

By the definition of symmetric functions, all parties have the same input domain. Recall that we have assumed that f is a non-constant symmetric function with minimal input domain. Also recall that f outputs a single bit.

We first prove the following lemma:

Lemma 3. *For all $P_i \in \{P_1, P_2, \ldots, P_n\}$, and for all $(\boldsymbol{x}, \boldsymbol{r})$ and $(\boldsymbol{x}', \boldsymbol{r}')$ such that $x_i \neq x'_i$,*

$$(M_{P_i}(\boldsymbol{x}, \boldsymbol{r}), f(\boldsymbol{x})) \neq (M_{P_i}(\boldsymbol{x}', \boldsymbol{r}'), f(\boldsymbol{x}'))$$

Proof. For the sake of contradiction, assume that this lemma is not true. Then there exists two executions, one with inputs $\boldsymbol{x}$ and random tapes $\boldsymbol{r}$ and the other one with inputs $\boldsymbol{x}'$ and random tapes $\boldsymbol{r}'$, such that $x_i \neq x'_i$ but

$$(M_{P_i}(\boldsymbol{x}, \boldsymbol{r}), f(\boldsymbol{x})) = (M_{P_i}(\boldsymbol{x}', \boldsymbol{r}'), f(\boldsymbol{x}'))$$

Since f has minimal input domain, f is sensitive to (P_i, x_i, x'_i), which means that there exists $\{\tilde{x}_j\}_{j \neq i}$ such that

$$f(\tilde{x}_1, \ldots, \tilde{x}_{i-1}, x_i, \tilde{x}_{i+1}, \ldots, \tilde{x}_n) \neq f(\tilde{x}_1, \ldots, \tilde{x}_{i-1}, x'_i, \tilde{x}_{i+1}, \ldots, \tilde{x}_n).$$

Let $\tilde{\boldsymbol{x}} = (\tilde{x}_1, \ldots, \tilde{x}_{i-1}, x_i, \tilde{x}_{i+1}, \ldots, \tilde{x}_n)$ and $\tilde{\boldsymbol{x}}' = (\tilde{x}_1, \ldots, \tilde{x}_{i-1}, x'_i, \tilde{x}_{i+1}, \ldots, \tilde{x}_n)$. Then $\tilde{x}_i = x_i$, $\tilde{x}'_i = x'_i$, $\tilde{x}_j = \tilde{x}'_j$ for all $j \neq i$, but $f(\tilde{\boldsymbol{x}}) \neq f(\tilde{\boldsymbol{x}}')$. Since f outputs a single bit, either $f(\boldsymbol{x}) = f(\boldsymbol{x}') = f(\tilde{\boldsymbol{x}})$ or $f(\boldsymbol{x}) = f(\boldsymbol{x}') = f(\tilde{\boldsymbol{x}}')$. Without loss of generality, assume that $f(\boldsymbol{x}) = f(\boldsymbol{x}') = f(\tilde{\boldsymbol{x}})$.

We first show that there exists $\tilde{\boldsymbol{r}}$ such that $(M_{P_i}(\boldsymbol{x}, \boldsymbol{r}), f(\boldsymbol{x})) = (M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}), f(\tilde{\boldsymbol{x}}))$. Since $\boldsymbol{x}, \tilde{\boldsymbol{x}}$ satisfy that $x_i = \tilde{x}_i$ and $f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}})$, by Property 1, the following two distributions are identical:

$$\{\texttt{View}(P_i, \boldsymbol{x})\} \equiv \{\texttt{View}(P_i, \tilde{\boldsymbol{x}})\}$$

Thus, there exists $\tilde{\boldsymbol{r}}$ such that $\mathtt{View}(P_i, \boldsymbol{x}, \boldsymbol{r}) = \mathtt{View}(P_i, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$. Since $(M_{P_i}(\boldsymbol{x}, \boldsymbol{r}), f(\boldsymbol{x}))$ is determined by P_i's view, we have $(M_{P_i}(\boldsymbol{x}, \boldsymbol{r}), f(\boldsymbol{x})) = (M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}), f(\tilde{\boldsymbol{x}}))$. Recall that $(M_{P_i}(\boldsymbol{x}, \boldsymbol{r}), f(\boldsymbol{x})) = (M_{P_i}(\boldsymbol{x}', \boldsymbol{r}'), f(\boldsymbol{x}'))$. Therefore, $(M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}), f(\tilde{\boldsymbol{x}})) = (M_{P_i}(\boldsymbol{x}', \boldsymbol{r}'), f(\boldsymbol{x}'))$.

Let $\tilde{\boldsymbol{r}}' = (\tilde{r}_1, \ldots, \tilde{r}_{i-1}, r'_i, \tilde{r}_{i+1}, \ldots, \tilde{r}_n)$, i.e., $\tilde{\boldsymbol{r}}' = \tilde{\boldsymbol{r}}$ except $\tilde{r}'_i = r'_i$. We will prove that $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}') = M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}) = M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$ by induction:

- Consider the first message in $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$. If it is a message sent from P_i to another party, then this message is fully determined by $\tilde{x}'_i = x'_i$ and $\tilde{r}'_i = r'_i$ since P_i does not receive any message from other parties. Thus, this message is identical to the first message in $M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$. Since $M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}) = M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$, the statement holds for the first message.
  If the first message in $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$ is received from another party, then this message is fully determined by $\{\tilde{x}'_j, \tilde{r}'_j\}_{j \neq i}$ since P_i does not send any message to other parties. Note that $\{\tilde{x}'_j, \tilde{r}'_j\}_{j \neq i} = \{\tilde{x}_j, \tilde{r}_j\}_{j \neq i}$. Thus this message is identical to the first message in $M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$. Since $M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}) = M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$, the statement holds for the first message.
- Assume the statement holds for the first $\ell - 1$ messages. For the ℓ-th message, if it is a message sent from P_i to another party, then this message is determined by $\tilde{x}'_i = x'_i, \tilde{r}'_i = r'_i$ and the first $\ell - 1$ messages in $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$. According to the induction hypothesis, the first $\ell - 1$ messages in $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$ are identical to the first $\ell - 1$ messages in $M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$. We also have $(\tilde{x}'_i, \tilde{r}'_i) = (x'_i, r'_i)$. Thus, the ℓ-th message in $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$ is identical to the ℓ-th message in $M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$ as well. Since $M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}) = M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$, the statement holds for the first ℓ messages.
  If the ℓ-th message of P_i is received from another party, then this message is fully determined by $\{\tilde{x}'_j, \tilde{r}'_j\}_{j \neq i}$ and the first $\ell - 1$ messages in $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$. According to the induction hypothesis, the first $\ell - 1$ messages in $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$ are identical to the first $\ell - 1$ messages in $M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$. We also have $\{\tilde{x}'_j, \tilde{r}'_j\}_{j \neq i} = \{\tilde{x}_j, \tilde{r}_j\}_{j \neq i}$. Thus, the ℓ-th message in $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$ is identical to the ℓ-th message in $M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$ as well. Since $M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}) = M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$, the statement holds for the first ℓ messages.
- Therefore, by induction, the statement holds for all ℓ. We have $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}') = M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}) = M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$.

Recall that $f(\tilde{\boldsymbol{x}}') \neq f(\tilde{\boldsymbol{x}})$. On the other hand, for parties in $\{P_j\}_{j \neq i}$, their views are determined by $\{\tilde{x}'_j, \tilde{r}'_j\}_{j \neq i}$ and $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$. Since $\{\tilde{x}'_j, \tilde{r}'_j\}_{j \neq i} = \{\tilde{x}_j, \tilde{r}_j\}_{j \neq i}$ and $M_{P_i}(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}') = M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$, parties in $\{P_j\}_{j \neq i, j \leq n}$ will obtain the same output in both the execution with $(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{r}}')$ and the execution with $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$, which contradicts with $f(\tilde{\boldsymbol{x}}') \neq f(\tilde{\boldsymbol{x}})$.

Lemma 3 shows that the messages a party (of the first n parties) receives or sends together with the output can determine his input. Without loss of generality, assume that $0, 1$ are in the input domain. Now consider the first t parties $P_1, P_2, \ldots, P_t$. For all $1 \leq i \leq t$, and for all vectors V subject to

$$\Pr[(\mathtt{View}(P_1, \boldsymbol{x}), \ldots, \mathtt{View}(P_{i-1}, \boldsymbol{x})) = V] \neq 0,$$

we define an encoding scheme $(\mathtt{Enc}, \mathtt{Dec})$ for the message space $\{0, 1\}$ as follows:

- Let $\boldsymbol{x} = (0, 0, ..., 0, 1)$ (i.e., all inputs are 0 except the last input is 1) and $\boldsymbol{x}' \in \{0,1\}^n$ subject to $x'_i = 1$ and $x'_j = 0$ for all $j \neq i$. Since f is a symmetric function, we have $f(\boldsymbol{x}) = f(\boldsymbol{x}')$ but $x_i \neq x'_i$.
  $\mathtt{Enc}(0)$ samples $\boldsymbol{r}$ uniformly subject to $\{\mathtt{View}(P_j, \boldsymbol{x}, \boldsymbol{r})\}_{j=1}^{i-1} = V$ and outputs $M_{P_i}(\boldsymbol{x}, \boldsymbol{r})$.
  $\mathtt{Enc}(1)$ samples $\boldsymbol{r}'$ uniformly subject to $\{\mathtt{View}(P_j, \boldsymbol{x}', \boldsymbol{r}')\}_{j=1}^{i-1} = V$ and outputs $M_{P_i}(\boldsymbol{x}', \boldsymbol{r}')$.
- The decoding algorithm takes as input a codeword $c = M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}})$, where $\tilde{\boldsymbol{x}} \in \{\boldsymbol{x}, \boldsymbol{x}'\}$. Recall that $f(\boldsymbol{x}) = f(\boldsymbol{x}')$. Therefore, $f(\tilde{\boldsymbol{x}}) = f(\boldsymbol{x}) = f(\boldsymbol{x}')$. According to Lemma 3, $(M_{P_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{r}}), f(\tilde{\boldsymbol{x}}))$ can determine the input $\tilde{x}_i$. $\mathtt{Dec}(c)$ outputs the input determined by $(c, f(\boldsymbol{x}))$.

We first show that $\mathtt{supp}(0)$ and $\mathtt{supp}(1)$ of the encoding scheme are not empty. It is sufficient to show that there exist $\boldsymbol{r}$ and $\boldsymbol{r}'$ such that

$$(\mathtt{View}(P_1, \boldsymbol{x}, \boldsymbol{r}), \ldots, \mathtt{View}(P_{i-1}, \boldsymbol{x}, \boldsymbol{r})) = V$$

and

$$(\mathtt{View}(P_1, \boldsymbol{x}', \boldsymbol{r}'), \ldots, \mathtt{View}(P_{i-1}, \boldsymbol{x}', \boldsymbol{r}')) = V.$$

Recall that V satisfies that $\Pr[(\mathtt{View}(P_1, \boldsymbol{x}), \ldots, \mathtt{View}(P_{i-1}, \boldsymbol{x})) = V] \neq 0$. Therefore, the existence of $\boldsymbol{r}$ follows. Recall that $f(\boldsymbol{x}) = f(\boldsymbol{x}')$ and $x_j = x'_j$ for all $j \in \{1, 2, \ldots, i-1\}$, by Property 1, we have

$$\{\mathtt{View}(P_1, \boldsymbol{x}), \ldots, \mathtt{View}(P_{i-1}, \boldsymbol{x})\} \equiv \{\mathtt{View}(P_1, \boldsymbol{x}'), \ldots, \mathtt{View}(P_{i-1}, \boldsymbol{x}')\}.$$

Thus,

$$\begin{aligned} &\Pr[(\mathtt{View}(P_1, \boldsymbol{x}), \ldots, \mathtt{View}(P_{i-1}, \boldsymbol{x})) = V] \\ = &\Pr[(\mathtt{View}(P_1, \boldsymbol{x}'), \ldots, \mathtt{View}(P_{i-1}, \boldsymbol{x}')) = V] \neq 0. \end{aligned}$$

The existence of $\boldsymbol{r}'$ follows. This implies that the encoding scheme $(\mathtt{Enc}, \mathtt{Dec})$ is well defined.

Lemma 4. *The encoding scheme* $(\mathtt{Enc}, \mathtt{Dec})$ *constructed above is* $(t - i + 1)$-*private.*

We refer the readers to the full version of this paper [20] for the proof of Lemma 4.

According to Lemma 2, $|\mathtt{supp}(0)| \geq 2^{t-i+1}$. That is, for inputs $\boldsymbol{x} = (0, 0, \ldots, 0, 1)$, when fixing the views of the first $i - 1$ parties, the view of the i-th party has at least 2^{t-i+1} different possibilities. Consider the joint view of the first t parties when the inputs are $\boldsymbol{x}$. It has at least $\prod_{i=1}^{t} 2^{t-i+1} = 2^{t(t+1)/2}$ different views. It implies that the number of random bits required by the protocol in the worst case is at least $t(t+1)/2 \geq t^2/2$. Therefore, the randomness complexity of the protocol is at least $t^2/2$.

Remark 1. We note that Theorem 2 holds even if the output is only given to a strict (nonempty) subset of the parties.

To see it, note that for Lemma 3, the statement holds for P_i as long as there is a party $P_j \neq P_i$ that receives the function output. Therefore, if there are at least two parties that receive the output, Lemma 3 holds. If only one party receives output, say P_n, then the statement holds for all parties other than P_n. Then in the rest of the proof, we can continue to focus on the number of views of the first t parties. With the same argument, we can show that the randomness complexity is at least $t^2/2$.

5 Explicit Construction of Zero Sharing Generators

In this section, we will give an explicit construction of a *linear* (multi-phase) t-resilient zero sharing generator by using a linear t-wise independent PRG in a black box way.

Theorem 3. *Let $\mathbb{G}$ be a finite Abelian group. Let p and $n_1, n_2, \ldots, n_p$ be positive integers, and $\mathtt{Tr}_1, \mathtt{Tr}_2, \ldots, \mathtt{Tr}_p$ be full binary trees such that $\mathtt{Tr}_j$ has n_j leaf nodes for all $j \in \{1, 2, \ldots, p\}$. For each tree $\mathtt{Tr}_j$, let $\mathcal{A}_j$ denote the access set determined by $\mathtt{Tr}_j$. Set $n = n_1 + n_2 + \ldots + n_p$, $\mathcal{A} = \bigcup_{j=1}^{p} \mathcal{A}_j$, and D to be the largest depth of $\mathtt{Tr}_1, \mathtt{Tr}_2, \ldots, \mathtt{Tr}_p$. Suppose $F : \mathbb{G}^m \rightarrow \mathbb{G}^n$ is a linear t-wise independent PRG. Then there exists an explicit linear multi-phase t-resilient zero sharing generator with respect to the access set $\mathcal{A}$ that uses $(D-1) \cdot m$ random group elements in $\mathbb{G}$.*

Proof. For every tree $\mathtt{Tr}_j$ and every node $v \in \mathtt{Tr}_j$, the depth of v is the length of the path towards the root of $\mathtt{Tr}_j$ plus 1. I.e., the root node of $\mathtt{Tr}_j$ has depth 1, the two children of the root node of $\mathtt{Tr}_j$ have depth 2, and so on. Note that leaf nodes of $\mathtt{Tr}_j$ do not necessarily have the same depth.

Let $\mathtt{Fr}$ denote the collection of the trees $\mathtt{Tr}_1, \mathtt{Tr}_2, \ldots, \mathtt{Tr}_p$. $\mathtt{Fr}$ is also referred to as a forest. Recall that $F : \mathbb{G}^m \rightarrow \mathbb{G}^n$ is a linear t-wise independent PRG. To construct a linear multi-phase t-resilient zero sharing generator G, we will assign to each node v in $\mathtt{Fr}$ a linear combination of the outputs of F, denoted by $\mathsf{val}(v)$, such that for all internal node v and its two children c_0, c_1, $\mathsf{val}(v) = \mathsf{val}(c_0) + \mathsf{val}(c_1)$. Then the values associated with the leaf nodes in $\mathtt{Fr}$ represent the output of G.

Explicit Construction of Linear Multi-Phase Zero Sharing Generator. The construction works as follows:

1. Let $\boldsymbol{u} = (\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(D-1)}) \in \mathbb{G}^{(D-1)\times m}$ be the input of G, where D is the largest depth of $\mathtt{Tr}_1, \mathtt{Tr}_2, \ldots, \mathtt{Tr}_p$.
2. For all root node $\mathtt{rt}_j$, we set $\mathsf{val}(\mathtt{rt}_j) = 0$.
3. From $d = 2$ to D, we will assign values to all nodes of depth d in $\mathtt{Fr}$. Let ℓ_d denote the number of nodes of depth d. Since $\mathtt{Tr}_1, \mathtt{Tr}_2, \ldots, \mathtt{Tr}_p$ are full binary trees, ℓ_d is even. We use $c_1, c_1, \ldots, c_{\ell_d}$ to denote the nodes of depth

d such that for all $i \in \{1, 2, \ldots, \ell_d/2\}$, (c_{2i-1}, c_{2i}) are the two children of a node v_i of depth $d-1$.
Suppose we have assigned values to all nodes of depth $d-1$ in $\mathtt{Fr}$. We compute $\boldsymbol{y}^{(d)} = F(\boldsymbol{u}^{(d-1)})$. Then for all $i \in \{1, 2, \ldots, \ell_d/2\}$, we set $\mathsf{val}(c_{2i-1}) = y_i^{(d)}$ and $\mathsf{val}(c_{2i}) = \mathsf{val}(v_i) - y_i^{(d)}$. In this way, for the node v_i and its two children c_{2i-1}, c_{2i}, we have $\mathsf{val}(v_i) = \mathsf{val}(c_{2i-1}) + \mathsf{val}(c_{2i})$.

4. The output of G are the values associated with the leaf nodes in $\mathtt{Fr}$. In particular, for all $j \in \{1, 2, \ldots, p\}$, $\boldsymbol{r}^{(j)} = (r_1^{(j)}, \ldots, r_{n_j}^{(j)})$ are the values associated with the leaf nodes of $\mathtt{Tr}_j$.

Lemma 5. *The above construction is a linear multi-phase t-resilient zero sharing generator.*

We refer the readers to the full version of this paper [20] for the proof of Lemma 5.

When $\mathbb{G}$ is a finite field $\mathbb{F}$, by Theorem 1, we can instantiate the linear t-wise independent PRG $F : \mathbb{F}^m \to \mathbb{F}^n$ with input size $m = O(t \cdot \log n)$. For all $j \in \{1, 2, \ldots, p\}$, we can use a full binary tree $\mathtt{Tr}_j$ with n_j leaf nodes of depth $O(\log n_j) = O(\log n)$. Thus, we have the following corollary.

Corollary 2. *Let $\mathbb{F}$ be a finite field. Let p and $n_1, n_2, \ldots, n_p$ be positive integers, and $\mathtt{Tr}_1, \mathtt{Tr}_2, \ldots, \mathtt{Tr}_p$ be full binary trees such that $\mathtt{Tr}_j$ has n_j leaf nodes of depth $O(\log n_j)$ for all $j \in \{1, 2, \ldots, p\}$. For each tree $\mathtt{Tr}_j$, let $\mathcal{A}_j$ denote the access set determined by $\mathtt{Tr}_j$. Set $n = n_1 + n_2 + \ldots + n_p$ and $\mathcal{A} = \bigcup_{j=1}^{p} \mathcal{A}_j$. Then there exists an explicit linear multi-phase t-resilient zero sharing generator that uses $O(t \cdot \log^2 n)$ random elements in $\mathbb{F}$.*

6 Upper Bound for Addition

In this section we prove our main new upper bounds, obtaining an explicit version of the previous upper bound for XOR from [27] and extending it to Abelian group addition. In the full version of this paper [20], we show (1) how to amortize randomness complexity over multiple executions, (2) how to construct an explicit protocol for any symmetric Boolean functions with $O(t^2 \cdot \log^3 n)$ random bits, and (3) how to construct an explicit protocol for general circuits with helper parties, which uses $O(t^2 \cdot \log s)$ random bits, where s is the circuit size.

We start by considering a function f that computes addition of n elements in a finite Abelian group $\mathbb{G}$. Concretely, f takes $x_i \in \mathbb{G}$ from the party P_i and computes $\sum_{i=1}^{n} x_i$. Assuming the existence of a linear t-resilient zero sharing generator $G : \mathbb{G}^m \to \mathbb{G}^n$, we construct an n-party computation protocol for f against t corrupted parties with perfect semi-honest security.

Theorem 4. *Let m, n, t be positive integers, $\mathtt{Tr}$ be a full binary tree with n leaf nodes, and $\mathbb{G}$ be a finite Abelian group. Let $f : \mathbb{G}^n \to \mathbb{G}$ be the addition function which is defined by $f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i$. Assume that $G : \mathbb{G}^m \to \mathbb{G}^n$ is a linear $(4t+1)$-resilient zero sharing generator with respect to the access set*

$\mathcal{A}$ determined by $\mathtt{Tr}$*. There is an n-party computation protocol for f against t corrupted parties with perfect semi-honest security, which uses $(t+1)\cdot m$ random group elements in $\mathbb{G}$.*

Proof. We first construct a protocol for f assuming the existence of an ideal functionality $\mathcal{F}_{\text{rand}}$ that distributes correlated randomness to all parties. For a full binary tree $\mathtt{Tr}$ with n leaf nodes, it has exactly $n-1$ internal nodes. We use $\{1,2,\ldots,n\}$ to label the leaf nodes in $\mathtt{Tr}$, and $\{n+1,n+2,\ldots,2n-1\}$ to label the internal nodes in $\mathtt{Tr}$. We also use $\mathtt{rt}$ to denote the root of $\mathtt{Tr}$.

Protocol with Ideal Functionality $\mathcal{F}_{rand}$. Consider an ideal functionality $\mathcal{F}_{\text{rand}}$ that samples $\boldsymbol{u}\in\mathbb{G}^m$ uniformly, computes $\boldsymbol{r}=(r_1,r_2,\ldots,r_n)=G(\boldsymbol{u})$, and distributes r_i to the party P_i for all $i\in\{1,2,\ldots,n\}$. All parties run the following steps:

1. Each party P_i locally computes $g_i=x_i+r_i$.
2. For each node v in $\mathtt{Tr}$, let S_v be the set of indices of leaf nodes that are descendants of v. We will ask a single party to compute $g_v:=\sum_{i\in S_v} g_i$. Note that for all leaf nodes $v\in\{1,2,\ldots,n\}$, we have already computed $g_v=x_v+r_v$ in Step 1. Now we describe how parties compute g_v for all internal nodes. Recall that $\mathtt{Tr}$ has $n-1$ internal nodes. From $i=1$ to $n-1$, all parties run the following steps:
   (a) Let v be the first internal node in $\mathtt{Tr}$ such that g_v has not been computed but g_{c_0},g_{c_1} have been computed, where c_0,c_1 are the two children of v. Suppose that g_{c_0} is computed by P_{j_0}, and g_{c_1} is computed by P_{j_1}.
   (b) P_i receives g_{c_0} from P_{j_0} and receives g_{c_1} from P_{j_1}. Then P_i computes $g_v=g_{c_0}+g_{c_1}$.
3. Note that in the last iteration of Step 2, P_{n-1} computes $g_{\mathtt{rt}}$ for the root node $\mathtt{rt}$. Then
$$g_{\mathtt{rt}}=\sum_{i=1}^{n} g_i=\sum_{i=1}^{n} x_i+\sum_{i=1}^{n} r_i.$$
Since G is a zero sharing generator and $\boldsymbol{r}=(r_1,r_2,\ldots,r_n)$ is the output of G, we have $\sum_{i=1}^{n} r_i=0$. Therefore, $g_{\mathtt{rt}}=\sum_{i=1}^{n} x_i$. Thus, P_{n-1} learns $f(\boldsymbol{x})$. P_{n-1} sends the result to all other parties.

The correctness of our construction follows from the description. we show that our construction is secure in the full version of this paper [20].

Realizing $\mathcal{F}_{rand}$. To obtain an n-party computation protocol for f in the plain model, it is sufficient to realize $\mathcal{F}_{\text{rand}}$. We simply follow the approach in [27]: Recall that G is a linear zero sharing generator. To realize $\mathcal{F}_{\text{rand}}$, we ask each party P_i of the first $t+1$ parties randomly samples $\boldsymbol{u}^{(i)}\in\mathbb{G}^m$, computes $\boldsymbol{r}^{(i)}=G(\boldsymbol{u}^{(i)})$, and distributes $r_j^{(i)}$ to P_j for all $j\neq i$. Then all parties locally set $\boldsymbol{r}=\boldsymbol{r}^{(1)}+\ldots+\boldsymbol{r}^{(t+1)}=G(\boldsymbol{u}^{(1)}+\ldots+\boldsymbol{u}^{(t+1)})$. The security follows from the fact that at least one of the first $t+1$ parties is not corrupted. Therefore,

$\boldsymbol{u} = \sum_{i=1}^{t+1} \boldsymbol{u}^{(i)}$ is uniformly random and $\boldsymbol{r} = G(\boldsymbol{u})$ has the same distribution as that generated by $\mathcal{F}_{\mathrm{rand}}$.

In summary, the whole protocol uses $(t+1) \cdot m$ random elements in $\mathbb{G}$.

When $\mathbb{G}$ is a finite field $\mathbb{F}$, and when we use a full binary tree `Tr` with n leaf nodes of depth $O(\log n)$, by Corollary 2, there is an explicit linear $(4t+1)$-resilient zero sharing generator $G : \mathbb{F}^m \to \mathbb{F}^n$ with input size $m = O(t \cdot \log^2 n)$. We have the following corollary.

Corollary 3. *Let n, t be positive integers, $\mathbb{F}$ be a finite field, and $f : \mathbb{F}^n \to \mathbb{F}$ be the addition function which is defined by $f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i$. There is an n-party computation protocol for f against t corrupted parties with perfect semi-honest security, which uses $O(t^2 \cdot \log^2 n)$ random field elements in $\mathbb{F}$.*

Acknowledgements. Y. Ishai was supported by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20. V. Goyal and Y. Song were supported by the NSF award 1916939, DARPA SIEVE program under Agreement No. HR00112020025, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award. Y. Song was also supported by a Cylab Presidential Fellowship.

References

1. Ananth, P., Ishai, Y., Sahai, A.: Private circuits: A modular approach. In: CRYPTO 2018. pp. 427–455 (2018)
2. Andrychowicz, M., Dziembowski, S., Faust, S.: Circuit compilers with o(1/log (n)) leakage rate. In: EUROCRYPT 2016. pp. 586–615 (2016)
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: ACM CCS 2016. pp. 116–129 (2016)
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC 1988. pp. 1–10 (1988)
5. Benaloh, J.C.: Secret sharing homomorphisms: Keeping shares of A secret sharing. In: CRYPTO '86. pp. 251–260 (1986)
6. Blundo, C., De Santis, A., Persiano, G., Vaccaro, U.: Randomness Complexity of Private Computation. Comput. Complex. **8**(2), 145168 (1999)
7. Blundo, C., Galdi, C., Persiano, G.: Low-randomness constant-round private XOR computations. Int. J. Inf. Sec. **6**(1), 15–26 (2007)
8. Bonawitz, K.A., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: CCS 2017. pp. 1175–1191 (2017)
9. Canetti, R., Kushilevitz, E., Ostrovsky, R., Rosén, A.: Randomness versus Fault-Tolerance. Journal of Cryptology **13**(1), 107–142 (2000)
10. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. J. Cryptol. **1**(1), 65–75 (1988)

11. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: STOC 1988. pp. 11–19 (1988)
12. Chor, B., Goldreich, O., Hasted, J., Freidmann, J., Rudich, S., Smolensky, R.: The bit extraction problem or t-resilient functions. In: FOCS 1985. pp. 396–407 (1985)
13. Chor, B., Kushilevitz, E.: A Communication-Privacy Tradeoff for Modular Addition. Inf. Process. Lett. **45**(4) (1993)
14. Coron, J.S., Greuet, A., Zeitoun, R.: Side-channel masking with pseudo-random generator. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 342–375. Springer International Publishing, Cham (2020)
15. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: USENIX NSDI 2017. pp. 259–282 (2017)
16. Data, D., Prabhakaran, V.M., Prabhakaran, M.M.: Communication and randomness lower bounds for secure computation. IEEE Trans. Inf. Theory **62**(7), 3901–3929 (2016)
17. Faust, S., Paglialonga, C., Schneider, T.: Amortizing randomness complexity in private circuits. In: ASIACRYPT 2017, Part I. pp. 781–810 (2017)
18. Gál, A., Rosén, A.: A theorem on sensitivity and applications in private computation. SIAM J. Comput. **31**(5), 1424–1437 (2002)
19. Goyal, V., Ishai, Y., Song, Y.: Private circuits withquasilinear randomness. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022. pp. 192–221. Springer International Publishing, Cham (2022)
20. Goyal, V., Ishai, Y., Song, Y.: Tight bounds on the randomness complexity of secure multiparty computation. Cryptology ePrint Archive, Paper 2022/799 (2022), `https://eprint.iacr.org/2022/799`, `https://eprint.iacr.org/2022/799`
21. Gl, A., Rosn, A.: $\Omega(\log n)$ Lower Bounds on the Amount of Randomness in 2-Private Computation. SIAM Journal on Computing **34**(4), 946–959 (2005), earlier version in STOC 2003
22. Ishai, Y., Kushilevitz, E., Li, X., Ostrovsky, R., Prabhakaran, M., Sahai, A., Zuckerman, D.: Robust pseudorandom generators. In: ICALP 2013. pp. 576–588 (2013)
23. Ishai, Y., Malkin, T., Strauss, M.J., Wright, R.N.: Private multiparty sampling and approximation of vector combinations. Theor. Comput. Sci. **410**(18), 1730–1745 (2009)
24. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. pp. 463–481 (2003)
25. Jakoby, A., Liskiewicz, M., Reischuk, R.: Private computations in networks: Topology versus randomness. In: STACS 2003. pp. 121–132 (2003)
26. Knuth, D., Yao, A.: Algorithms and Complexity: New Directions and Recent Results, chap. The complexity of nonuniform random number generation. Academic Press (1976)
27. Kushilevitz, E., Mansour, Y.: Randomness in Private Computations. SIAM Journal on Discrete Mathematics **10**(4), 647–661 (1997), earlier version in PODC 1996
28. Kushilevitz, E., Ostrovsky, R., Prouff, E., Rosén, A., Thillard, A., Vergnaud, D.: Lower and upper bounds on the randomness complexity of private computations of AND. SIAM J. Discret. Math. **35**(1), 465–484 (2021), earlier version in TCC 2019
29. Kushilevitz, E., Ostrovsky, R., Rosén, A.: Characterizing Linear Size Circuits in Terms of Privacy. In: STOC 1996. p. 541550 (1996)
30. Kushilevitz, E., Ostrovsky, R., Rosén, A.: Amortizing randomness in private multiparty computations. SIAM J. Discret. Math. **16**(4), 533–544 (2003)
31. Kushilevitz, E., Rosén, A.: A Randomness-Rounds Tradeoff in Private Computation. SIAM J. Discret. Math. **11**(1), 6180 (feb 1998)