Scalable Multiparty Garbling

Gabrielle Beck
Johns Hopkins University
Baltimore, USA
becgabri@cs.jhu.edu

Aarushi Goel
NTT Research
Sunnyvale, USA
aarushi.goel@ntt-research.com

Aditya Hegde
Johns Hopkins University
Baltimore, USA
ahegde@cs.jhu.edu

Abhishek Jain
Johns Hopkins University and NTT Research
Baltimore and Sunnyvale, USA
abhishek@cs.jhu.edu

Zhengzhong Jin
Massachusetts Institute of Technology
Cambridge, USA
zzjin@mit.edu

Gabriel Kaptchuk
Boston University
Boston, USA
kaptchuk@bu.edu

ABSTRACT

Multiparty garbling is the most popular approach for constant-round secure multiparty computation (MPC). Despite being the focus of significant research effort, instantiating prior approaches to multiparty garbling results in constant-round MPC that can not realistically accommodate large numbers of parties. In this work we present the first global-scale multiparty garbling protocol. The per-party communication complexity of our protocol *decreases* as the number of parties participating in the protocol increases—for the first time matching the asymptotic communication complexity of non-constant round MPC protocols. Our protocol achieves malicious security in the *honest-majority* setting and relies on the hardness of the Learning Party with Noise assumption.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

mpc, garbling, packed secret sharing, constant rounds

ACM Reference Format:
Gabrielle Beck, Aarushi Goel, Aditya Hegde, Abhishek Jain, Zhengzhong Jin, and Gabriel Kaptchuk. 2023. Scalable Multiparty Garbling. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623132

1 INTRODUCTION

Secure multiparty computation (MPC) [20, 28, 50, 86] is a class of cryptographic protocols that allows mutually distrusting parties to compute a function over hidden inputs. Since the eighties—when the first feasibility results were established—continuous progress has been made towards improving the efficiency of MPC protocols along various dimensions. Such improvements have resulted in the creation of toolchains for MPC (e.g., [14, 39, 57, 72, 73]) that are concretely efficiency for some limited applications; as a result, MPC has been deployed in industry [24, 82], government [6], and for social good [66, 80].

Global-Scale MPC. Although enthusiasm for MPC is growing, the ability to deploy MPC is hampered by existing protocol's lack of scalability. Existing deployments have been forced to use only a few computational parties co-located in the same geographical area in an effort to reduce latency. While these deployment choices make current-generation MPC protocols concretely efficient, they make it harder to believe the non-collusion assumptions required for maintaining privacy. Specifically, it may be feasible for an attacker to convince a small number of parties into releasing their view of the protocol, compromising the confidentiality of party's inputs. Co-location either requires the use of cloud computing resources (introducing another attack surface) or parties that are *already* geographically close to one another, increasing the chances that they have some preexisting relationship. To mitigate these problems, there is a need for MPC protocols that *scale*, both in terms of the number of parties, and the robustness to geographical diversity of those parties.

Due to a significant line of recent work [12, 31, 42, 52, 53, 55, 83], we now know of protocols that scale gracefully as the number of parties increases—state-of-the-art protocols have the communication complexity *independent* of the number of parties and are concretely efficient [42, 53]. Generally, these protocols dictate that parties jointly compute circuits in a gate-by-gate fashion, meaning that the computation requires communication rounds proportional to the *depth of the circuit* being computed. Unfortunately, network latency between parties is a key determinant of the protocol runtime in gate-by-gate protocols (see, e.g. [85] for a discussion), so even these state-of-the-art protocols fall short when protocol participants are globally distributed.

Constant round MPC protocols [11, 16, 18, 32, 46, 58–60, 64, 68, 76, 85, 86] are more appropriate for high latency settings, as the number of times parties must communicate is independent of the circuit size. There are two well-studied approaches to constant-round MPC—one based on fully-homomorphic encryption [49] and another based on garbled circuits [11, 86]. The latter has been studied more extensively because of its better potential for efficient solutions (despite incurring asymptotically worse communication).

The second approach follows a template first proposed by Beaver, Micali, and Rogaway (BMR) [11]: the parties first execute a *garbling phase*, in which they jointly compute a garbled circuit of the desired

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3623132

functionality within an MPC protocol. The garbling phase is then followed by an *output evaluation phase*, in which the parties exchange inputs and evaluate the garbled circuit. Since garbled gates can be computed in parallel, the resulting protocol has constant rounds. Throughout this work we refer to the process of jointly computing the garbled circuit as *multiparty garbling*.

Barriers to Efficient, Scalable Multiparty Garbling. Although multiparty garbling is a well-studied approach for constant-round MPC, existing proposals cannot realistically be used to perform global-scale computations. Asymptotically efficient constructions of the BMR template can be obtained by making non-black-box use of cryptography used during garbling [11], but representing the cryptography as circuitry introduces prohibitive overheads.

The best known black-box multiparty garbling protocols, on the other hand, require per-gate total communication (and computation) that is *quadratic in the number of parties* [16, 18, 32, 58–60, 68, 85], meaning these protocols scale poorly with the number of parties. As is common in the literature on efficient MPC, these works split the garbling phase into a circuit independent *pre-processing phase* and a circuit dependent *garbling phase*. Recent works [16, 18] have demonstrated methods that reduce the complexity of the circuit-dependent garbling phase and output evaluation phase to be linear in the number of parties, but still require a pre-processing phase with quadratic complexity.

Thus, overall, the quadratic barrier stands for efficient multiparty garbling—in both the honest and dishonest majority settings. This poses a major barrier for global-scale computations; the combination of the quadratic dependence on the number of parties and the fact that garbling inevitably increases the circuit size by a multiplicative factor dependent on the security parameter results in impractical solutions, even for moderately-sized circuits. We therefore ask the following question:

Can we design an efficient and scalable multiparty garbling protocol?

We answer this question in the affirmative, taking a significant step towards efficient, global-scale, constant-round MPC.

1.1 Our Contributions

We present a new *scalable* constant-round multiparty garbling protocol for boolean circuits in the honest-majority setting, where the total per-party communication complexity (see below) in our protocol *decreases* as the number of parties increase. To design this protocol, we combine several recent advances in efficient MPC and carefully compose them using bespoke subprotocols. In more detail, our protocol has the following features:

- **Communication Complexity:** The total communication complexity of the protocol is independent of the number of parties (i.e., is $O(|C|)$,[1] where C is the circuit being computed), meaning that the per-party communication actually *decreases* as the number of parties increases. Similar to prior constant round MPC protocols [11, 16, 18, 32, 58–60, 68, 76, 85], our protocol utilizes a single round of broadcast to reconstruct the circuit to all parties, but otherwise runs over point-to-point channels.[2]
- **Computation Complexity:** The per-party computation complexity of the protocol is independent of the number of parties (i.e., the total computation complexity of the protocol is $O(n|C|)$). This computational complexity is inherent in the constant-round BMR template, as each party needs to *evaluate* the garbled circuit.
- **Security and Assumptions:** Our protocol achieves *malicious security* against $t < \frac{n-2\ell+1}{2}$ corrupt parties, where $\ell \geq 1$ is a tunable parameter induced by the use of packed secret sharing[3]. The security of our construction relies on the Learning Parity with Noise over Large Fields (LPN) assumption. The use of LPN in our construction demonstrates that "less-powerful" assumptions (i.e., ones that are not known to imply FHE) are sufficient for designing efficient and scalable constant round MPC.

Our protocol is the first *constant-round MPC* to asymptotically match the best known communication complexity of concretely efficient, gate-by-gate MPC protocols without using fully-homomorphic encryption.[4] As such, our protocol demonstrates a path towards practical MPC in high-latency settings, where gate-by-gate protocols typically struggle.

Our Techniques. We use the following techniques to achieve our result:

- **Generic Approach for Honest Majority.** We first identify an approach for scalable multiparty garbling in the honest majority setting. In this approach, we rely on encryption schemes where given shares of the key, message and randomness, it is possible to non-interactively obtain shares of the corresponding ciphertext. Such encryption schemes have been used in the recent works by Ben-Efraim et al. [16, 18] in order to optimize the efficiency of the output evaluation phase. Our main observation is that this approach can be successfully "married" with packed secret-sharing (and other) techniques including ones from a recent work by Goyal et al. [54] to compute garbled circuits with $O(|C|)$ total communication.
- **Instantiation.** We instantiate our approach with an encryption scheme based on the learning parity with noise (LPN) assumption over large fields. This leads to unique challenges in the honest majority setting. In particular, we custom design efficient subprotocols that enable distributed generation of the cryptographic material used in this encryption scheme. Finally, we show how to augment the above approach using known techniques [12, 31, 38, 54] to achieve malicious security.

Evaluation and Analysis. To evaluate the efficiency of our construction, we (1) implement the semi-honest secure version of our protocol and use it to evaluate popular MPC benchmark circuits, and (2) programmatically estimate the concrete computation and communication costs of our maliciously secure protocol.

Our benchmarks of the semi-honest secure version of our protocol indicates that it is practical and scalable. Garbling the AES-128

[1] The $O(\cdot)$ notation suppresses linear terms in the security parameter and other logarithmic terms (independent of circuit size).

[2] We note that because the broadcast channel is used only in the final round, this broadcast channel is particularly well suited to implementation via a website, where parties can post their shares and then download the garbled circuit at a later time.

[3] ℓ corresponds to the number of secrets packed into one share. We refer the reader to Section 2 for more details.

[4] Up to a security parameter factor, introduced from encrypting each gate.

circuit takes around 126s even when 512 parties participate in the protocol, with the circuit dependent phase constituting just 15.5s. Moreover, the runtimes of the protocol does not seem to vary significantly with the number of parties and depends mainly on the size of the circuit being evaluated. Our analysis suggests that our protocol outperforms prior works on multiparty garbling in the honest majority setting [19], especially when run with a large number of parties.

Our estimates for the computation and communication costs of the maliciously secure protocol also suggest that the runtime of the protocol is practical and mainly depends on the size of the circuit being evaluated and does not vary significantly with the number of parties. We compare the performance of our protocols with prior works on efficient, maliciously secure, multiparty garbling [16, 85]. While prior works suffer from higher asymptotic overhead, our analysis indicates that even the concrete communication cost of our protocol is lower than those of prior works for $n \geq 350$. In this setting, our protocol achieves the lowest communication cost compared to existing solutions for multiparty garbling. Moreover, since our benchmarks and estimates suggests that runtimes are mostly independent of the number of parties, we believe our approach provides a viable solution for large scale computations with many participants.

2 PRELIMINARIES

We use $\mathbf{x}$ to denote a vector and $(\mathbf{x})_i$ to denote the i-th element in the vector. We use $[a, b]$ where $a \leq b$ to denote the set of integers $\{a, a+1, \ldots, b\}$. In this work, we design an honest majority MPC protocol that achieves security with abort against an static adversary in the client-server model. We provide formal definitions of this model in the full version. Let n denote the number of servers (which we call parties, for simplicity) and t denote the number of parties the adversary can corrupt. We assume that the parties have access to both point-to-point private and authenticated synchronous channels and a public synchronous broadcast channel, each of which has "unit" cost.

Secret Sharing. In this work we use the packed Shamir secret sharing scheme introduced by Franklin and Yung [44], a generalization of the Shamir secret sharing scheme [81] where each share corresponds to ℓ secrets. We denote a degree d Shamir sharing of a value x as $[x]_{\mathsf{d}}$. Building on an approach by Goyal et al. [54], We denote a degree-d packed Shamir sharing of a vector $\mathbf{x}$ as $[\mathbf{x}\,|\mathrm{pos}]_{\mathsf{d}}$, where pos is a set of positions in which the values of $\mathbf{x}$ are stored; when pos takes on a default value, we may just write $[\mathbf{x}]$. We refer the reader to [54] for more details and defer a complete description of the scheme to the full version.

Error Correcting Codes. Let Q, L, d, q be integers. An $[Q, L, d]_q$ error correcting code is a pair of algorithms $\mathbf{ECC} = (\mathsf{Enc}, \mathsf{Dec})$, where the encoding algorithm Enc takes a message $\mathbf{m} \in [1, q]^L$ as input, and outputs a codeword in $[1, q]^Q$. The decoding algorithm Dec takes a potentially corrupted codeword as input, and recovers the message. The distance of the code is the minimum Hamming distance between any two different codewords. We now discuss two properties of error correcting codes required in the construction of our protocol. We defer the proof of these properties to the full version.

Theorem 2.1. *Let $C = \{\mathbf{c} \mid \mathbf{c} = \mathbf{G}\cdot\mathbf{m}\} \subseteq \mathbb{F}^Q$ be an $[Q, L, d]$-binary linear code with generating matrix $\mathbf{G} \in \mathbb{F}^{Q\times L}$, then any sub-matrices consisted of $(Q - d + 1)$-rows of $\mathbf{G}$ is full rank.*

Theorem 2.2. *Let C be a code with parameters $[Q, L, d]_q$ and C' be another code with parameters $[Q', L', d']_{q'}$, where $q = 2^{L'}$, then the concatenated code $C \circ C'$ has parameters $[Q \cdot Q', L \cdot L']_{q'}$ with distance at least $d \cdot d'$.*

LPN Assumption and LPN Based Encryption. Our protocols rely on the Learning Parity with Noise (LPN) assumption over large fields that has also been used in a number of prior works [2, 3, 25, 62, 63]. We use a variation of the assumption stated by Boyle et al. [25]. In short, this assumption holds if it is difficult to distinguish the tuple $(\mathbf{A}, \mathbf{A}\cdot\mathbf{s} + \mathbf{e})$ and a random tuple, where $\mathbf{A}, \mathbf{s}$ are sampled from the uniform distribution and $\mathbf{e}$ is a noise vector with small Hamming weight sampled from $\mathsf{Ber}_{\tau_{\mathsf{lpn}}}$. A formal description of the assumption will be provided in the full version.

Given a $[Q_{\mathrm{ecc}}, L_{\mathrm{ecc}}, d]_2$ error-correcting code, we construct an CPA-secure encryption scheme under the LPN assumption with message space $\mathcal{M} = \mathbb{F}^{L_{\mathrm{ecc}}}$ as follows:

- $\mathsf{LPN.Keygen}(1^{\kappa_c})$: Sample $\mathbf{s} \leftarrow \mathbb{F}^{L_{\mathsf{lpn}}}$ uniformly at random.
- $\mathsf{LPN.Enc}(\mathbf{x}, \mathbf{s})$: To encrypt a message $\mathbf{x}$ under the key $\mathbf{s}$, first sample $\mathbf{A} \leftarrow \mathbb{F}^{L_{\mathsf{lpn}}\times Q_{\mathrm{ecc}}}$ uniformly at random and sample $\boldsymbol{\epsilon} \leftarrow \mathsf{Ber}_{\tau_{\mathsf{lpn}}}^{Q_{\mathrm{ecc}}}$. Output $(\mathbf{A}, \mathbf{s}\cdot\mathbf{A} + \boldsymbol{\epsilon} + \mathsf{ECC.Enc}(\mathbf{x}))$ as the ciphertext.
- $\mathsf{LPN.Dec}((\mathbf{A}, \mathbf{c}), \mathbf{s})$: Decrypt the ciphertext $\mathbf{c}$ using $\mathbf{s}$ by computing $\mathsf{ECC.Dec}(\mathbf{c} - \mathbf{s}\cdot\mathbf{A})$.

Note that security follows from the fact that $(\mathbf{A}, \mathbf{s}\cdot\mathbf{A} + \boldsymbol{\epsilon} + \mathsf{ECC.Enc}(\mathbf{x}))$ is computationally indistinguishable from $(\mathbf{A}, \mathbf{r} + \mathsf{ECC.Enc}(\mathbf{x}))$ where $\mathbf{r} \leftarrow \mathbb{F}^{Q_{\mathrm{ecc}}}$ is sampled uniformly at random. Correctness holds because $\mathbf{c} - \mathbf{s}\cdot\mathbf{A} = \boldsymbol{\epsilon} + \mathsf{ECC.Enc}(\mathbf{x})$ and if the Hamming weight of the noise $\boldsymbol{\epsilon}$ is no greater than $\lfloor (d-1)/2 \rfloor$, then the error correcting code can decrypt the message correctly.

3 BACKGROUND

BMR Constant Round Protocol Template. In their seminal work, Beaver, Micali, and Rogaway (BMR) [11] outline a template for constructing constant round MPC. The parties first perform a *garbling phase* by taking a generic (*i.e.* non-constant round) MPC protocol and using it to compute a garbled circuit of the functionality—rather than computing the function itself. The parties then initiate an *output evaluation phase*, in which they locally evaluate the garbled circuit to recover the function output. Because the garbling procedure is not inherently sequential, the tables can all be computed in parallel. Since computing each garbled table can be done in a constant number of rounds (by using an appropriate encryption scheme), the resulting protocol is itself constant round. Since the introduction of the BMR template, a long sequence of works (e.g., [16, 18, 32, 58–60, 68, 76, 85]) have investigated the efficiency of running protocols within the BMR template, leading to several improvements.

Black-Box Use of Cryptography. Beaver, Micali, and Rogaway's initial protocol proposed making non-black box use of the garbling algorithm of the garbled circuit scheme. For a gate g with input

wires a, b and output wire c computing the function $f : \{0,1\}^2 \to \{0,1\}$, the row of the garbled table corresponding to inputs $\alpha, \beta \in \{0,1\}$ is computed as

$$ct_{\alpha,\beta} = \mathsf{PRF}_{\mathsf{k}_{a,\alpha}}(g) \oplus \mathsf{PRF}_{\mathsf{k}_{b,\beta}}(g) \oplus \mathsf{k}_{c,f(\alpha,\beta)}$$

where $\mathsf{k}_{a,\alpha}$ is a random key/label associated with the wire a and the value $\alpha \in \{0,1\}$. While conceptually simple, the explicit circuit representation of PRF will be massive, resulting in an inefficient concrete construction. To obtain an efficient *black-box* solution, the PRF must somehow be evaluated *outside* the MPC protocol without compromising privacy or correctness.

Damgård and Ishai [32] devised an intuitive way to accomplish this goal: each party $\mathsf{P}_m \in \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ independently samples a pair of labels $(\mathsf{k}^m_{c,0}, \mathsf{k}^m_{c,1})$ for each wire c in the circuit. The parties then combine these independent labels into a single label containing n keys within the MPC protocol as

$$(\mathsf{k}^1_{c,0} \| \ldots \| \mathsf{k}^n_{c,0}, \mathsf{k}^1_{c,1} \| \ldots \| \mathsf{k}^n_{c,1}).$$

The parties then encrypt the combined labels by feeding locally expanded PRFs into the MPC. Thus the garbled table for gate g with input wires a, b and output wire c implementing the function f is computed as follows for inputs $\alpha, \beta \in \{0,1\}$ and party index $j \in [n]$: [5]

$$ct^j_{\alpha,\beta} = \bigoplus_{m=1}^{n} \mathsf{PRF}_{\mathsf{k}^m_{a,\alpha}}(g \| j) \oplus \bigoplus_{m=1}^{n} \mathsf{PRF}_{\mathsf{k}^m_{b,\beta}}(g \| j) \oplus \mathsf{k}^j_{c,f(\alpha,\beta)}$$

In other words, the garbled table for each gate consists of $4n$ ciphertexts, where each ciphertext is computed using n keys. Subsequent to the initial presentation of this protocol [32], Lindell et al. [68] showed that this approach can be extended to the malicious security case by incorporating simple local checks performed by the parties during the output evaluation phase.

Permuting Ciphertexts. There is one significant element of the BMR template that we have so far not addressed: the rows of each garbled table must be permuted so that its position in the table reveals nothing about its value. In practice, most prior works do this by sampling a random "mask" bit $\lambda_c \in \{0,1\}$ associated with each wire c. These bits are then used to select the order of the permutation of the table within the MPC.

Barriers to Scalable Multiparty Garbling. Existing techniques discussed so far, that make black-box use of cryptography are inherently not scalable. Since *each party contributes to encrypting every other party's key*, the circuit representation of garbling each gate is of size $O(n^2)$.[6] Thus, when running the garbling phase, the overall communication complexity will be at least quadratic in n (which clearly doesn't scale well as n grows)—no matter the efficiency of the MPC protocol used. Thus, reducing the size of the garbled tables is a necessary condition for scalable multiparty garbling.

[5] In the technical overview, we don't explicitly discuss how rows in the garbled table are permuted. We do this using standard point-and-permute [11] techniques by sampling random bit-masks for every wire in the circuit.

[6] Throughout the technical overview, we will generally omit the security parameter from our asymptotic notation, as our focus is the dependence on the number of parties.

Reducing the Number of Ciphertexts. Motivated by the desire to optimize the output evaluation phase of multiparty garbling, Ben-Efraim et al. [18] demonstrate a method in the dishonest majority setting for computing garbled tables with a *constant* number of ciphertexts without relying on non-black box use of cryptography. At the heart of their approach is a PRF that has key homomorphism [22]. If each party has a share of the wire key, encryption can be computed by (1) parties locally evaluating the PRF on their shares of the key, and (2) homomorphically combining the PRF outputs.

4 SCALABLE MULTIPARTY GARBLING IN THE HONEST MAJORITY SETTING

Linearly key-homomorphic PRFs are presently only known based on the Decisional Diffie-Hellman assumption in the random oracle model [22]. Ben Efraim et al. [16, 18] devised a way around the lack of key-homomorphic PRFs by using ring LWE and LPN (over boolean field) based encryption scheme (that allow parties to locally compute shares of the ciphertext, given shares of key, message and randomness), instead of a linearly key-homomorphic pseudo-random function. We build upon this approach to achieve better efficiency guarantees in the honest majority setting. When switching to the honest majority setting, we can leverage *threshold* secret sharing instead of additive secret sharing.

Following the above general approach of Ben-Efraim et al., we want parties to be able to locally compute *shares* of a single ciphertext, such that the ciphertext can be reconstructed during the output evaluation phase. Specifically, let the encryption scheme be such that $[ct] = \mathsf{ENC}([\mathsf{key}], [\mathsf{msg}]; [\mathsf{rand}])$. Further, let us imagine that for each gate g computing the function f, in addition to holding secret shares of the keys corresponding to the input wires a, b and output wire c, the parties also hold secret shares of some randomness for the encryption scheme. We want each party P_m to locally compute its share of the row $\alpha, \beta \in \{0,1\}$ as

$$[ct_{\alpha,\beta}]_m = \mathsf{ENC}\left(\left([\mathsf{k}_{a,\alpha}]_m \oplus [\mathsf{k}_{b,\beta}]_m\right), [\mathsf{k}_{c,f(\alpha,\beta)}]_m; [r_{\alpha,\beta}]_m\right).$$

Here $[\mathsf{k}_{a,\alpha}]_m \oplus [\mathsf{k}_{b,\beta}]_m$ correspond to shares of the key used to encrypt message $\mathsf{k}_{c,f(\alpha,\beta)}$ using randomness $r_{\alpha,\beta}$.

Roadmap Ahead. Given the discussion so far, we use the following roadmap to achieve our results.

- **Using an appropriate encryption scheme:** As mentioned earlier, Ben Efraim et al. [16, 18] observe that ring-LWE and LPN based encryption schemes satisfy the above properties and demonstrate how they can be used in the dishonest majority setting. These properties are also satisfied over threshold secret shares. In Section 5.1, we discuss which encryption scheme works better for us in the honest majority setting. Moreover, our choice of encryption scheme dictates the distribution from which the keys and randomness are sampled. In Section 5, we also discuss how to obtain threshold secret shares of the keys and randomness sampled from the appropriate distribution.
- **Achieving malicious security:** As observed in prior work, achieving malicious security in multiparty garbling involves protecting against two types of attacks: (1) malicious adversaries manipulating the MPC protocol used to compute the garbled

circuit, and (2) malicious adversaries injecting errors into the garbled circuit by using *inconsistent inputs*.

Towards addressing the first type of attack, we note that a long history of active research on honest majority malicious security compilers [31, 37, 45, 47, 48, 56, 67, 77] has significantly reduced the overhead of malicious security. In principle, these techniques can be lifted into the multiparty garbling setting; we note that when we instantiate our protocol, adapting these techniques will require some care, which we discuss in more detail in Section 5.5.

Let us now discuss defense against the second type of attacks. We note that in the honest majority regime, when using an encryption scheme with the above property, it is not actually possible for the malicious players to manipulate the value of the ciphertext directly, as the ciphertext within a threshold secret sharing is uniquely defined by the honest parties' shares. Indeed, this is much more of an immediate problem in the *dishonest majority* setting, where additive shares are more commonly used. However, we need to ensure that the keys and randomness used in the encryption are sampled from the correct distribution by preventing the adversary from influencing the sampling process. Fortunately, we observe that our approach for handling the first type of attacks can also be used to counter this attack. We defer discussion on how to address these attacks in Section 5.5.

- **MPC with $O(|\mathsf{C}|)$ communication:** With a garbled circuit with gate representations that are constant in the number of parties, the question is what protocol should the parties use to create the garbled circuit. Thankfully, MPC protocols with $O(|\mathsf{C}|)$ total communication have been the subject of significant research efforts [12, 33, 34, 47, 52–54]. All of these protocols rely heavily on *threshold packed secret sharing schemes* [44], a "Single-Instruction-Multiple-Data" (SIMD) version of threshold secret sharing schemes [81]. By operating on $O(n)$ elements at a time and using efficient multiplication protocols (e.g. [36]), these protocols are able to achieve total communication complexity independent of the number of parties. In particular, we rely on the techniques from a recent work by Goyal et al. [54], which is an efficient, non-constant round MPC for general circuits.

5 INSTANTIATING OUR APPROACH

With the overview outlined above acting as a clear roadmap to constructing scalable multiparty garbling scheme, we now explore how we can *instantiate* the necessary primitives and subprotocols.

5.1 Choice of Encryption Scheme

LWE and LPN based secret-key encryption schemes are of the form $\mathbf{k} \cdot \mathbf{A} + \mathbf{e} + \mathcal{L}(\mathbf{m})$, where $\mathbf{k}$ is the key vector, $\mathbf{A}$ is a public matrix, $\mathbf{e}$ is the random error vector, $\mathbf{m}$ is the message vector and $\mathcal{L}$ is a public linear function. Since, $\mathbf{A}$, $\mathcal{L}$ is public, computing the ciphertext only requires linear operations over the key vector $\mathbf{k}$, message vector $\mathbf{m}$ and the error vector $\mathbf{e}$. This essentially implies that if the parties hold shares of $\mathbf{k}$, $\mathbf{e}$ and the message $\mathbf{m}$, they can locally compute shares of the corresponding ciphertext without interaction. We note that, depending on the maximum fan-out $\mathsf{fanout}_{\mathsf{max}}$ across all gates in a given circuit, the number of unique $\mathbf{A}$ matrices that we require in general is $8 \times \mathsf{fanout}_{\mathsf{max}}$ [18]. Since these are public matrices, we can generate them *a priori*.

Compatibility between the encryption scheme that we use and known efficient techniques for honest majority MPC will dictate the overall efficiency and scalability of our multiparty garbling scheme. In instantiating this template, we will use use the straightforward equivalent of this encryption scheme based on LPN over a large field (similar assumptions have been used in several recent works [25, 41, 63]). To justify these choices, we briefly discuss the alternative assumptions that we could use—LWE and boolean LPN—and demonstrate why LPN over large fields is the most appropriate choice for our application.

LWE vs LPN. Several prior works prefer LPN over LWE for efficiency reasons. Unlike LPN, LWE is known to imply FHE and is believed to be a "more-powerful" assumption than LPN. As a result, in general, parameter sizes in LWE tend to be larger than the ones required in LPN. Moreover, the matrix $\mathbf{A}$ in LPN is the generator matrix corresponding to a probabilistic code generation algorithm. It is possible to choose matrices, where each column contains a small (constant) number of random non-zero coordinates, without weakening the security of LPN [1, 3]. Using such a matrix, computing $\mathbf{k} \cdot \mathbf{A}$ for any vector $\mathbf{k}$ can be done in time linear in the length of $\mathbf{k}$. On the other hand, to the best of our knowledge, no such optimizations are known in LWE and hence computing $\mathbf{k} \cdot \mathbf{A}$ requires time quadratic in the length of $\mathbf{k}$. As such, we can believe that LPN poses a more fruitful direction for instantiating our template.

LPN over a boolean field vs. LPN over a larger field. The LPN-based encryption of a message $\mathbf{m}$ requires encoding $\mathbf{m}$ using a linear error correcting code ECC.Enc, and adding the result to the output of the random function $\mathbf{k} \cdot \mathbf{A} + \mathbf{e}$. As such, the size of an LPN ciphertext depends on both (1) the efficiency of existing ECC.Enc, and (2) the best known attacks on the LPN assumption. LPN over large fields outperforms LPN over boolean fields in both criteria: (1) ECC.Enc's in larger fields tend to have better rates than binary ECC.Enc, and (2) in the large field setting, there exist variants of the LPN assumption (see [25, 43] for a detailed discussion) where the best known attack remains the same as in the boolean regime.

As a result, LPN over large fields provides equivalent levels of security with smaller parameter sizes. We note that this tradeoff is not always absolute: while LPN over larger fields might admit shorter vectors $\mathbf{k}$, $\mathbf{e}$ and a smaller matrix $\mathbf{A}$, representing each element requires multiple bits, which could result in the total representation that is larger than the equivalent construction from LPN over a boolean field.

We observe that in our setting it is still less efficient to use LPN over a boolean field because the parties run an MPC protocol to generate and use the cryptographic key material. Recall that since we rely on techniques from [54], we need to work in a field of size $O(|\mathsf{C}|)$. Thus, the parties will have to use a larger field *irrespective of our choice of the cryptographic assumption.* As such, LPN over a boolean field becomes *wasteful* in the context of our protocol—each bit will be represented in a large field anyhow—undermining the potential advantage of working with LPN over a boolean fields.

5.2 Sub-Protocol for Generating Errors

The errors in our LPN-based encryption are sampled from a Bernoulli distribution over $\mathbb{F}$, i.e., every element of the error vector is a random non-zero element in $\mathbb{F}$ with probability $\frac{1}{p}$ and zero with probability $1 - \frac{1}{p}$, where p is derived from the parameter choices. While efficient distributed protocols for generating shares of uniform random values in the field are known due to Damgård et al. [36] and Beerliova-Trubiniova et al. [13], to our knowledge, no such protocols are known for generating shares of values from this *biased distribution*.

To generate shares of biased bits, we use the following observation. Let us assume that p is a power of 2, i.e., of the form $p = 2^{\tau_{\mathsf{lpn}}}$. It is now easy to see that the product of τ_{lpn} random bits will be 1 with probability $1/p$ and 0 with probability $(p-1)/p$. To implement this idea, the parties can use our random bit sharing protocol (described below in Section 5.3) to sample shares of τ_{lpn} random bits and then multiply them to get a sharing of the appropriately-biased bit. If τ_{lpn} is constant, these multiplications can be done in a constant number of rounds. Moreover, to ensure that our total communication is $O(|\mathsf{C}|)$, we generate these shares in packed secret sharing form. We choose our LPN parameters to ensure that p is a power of 2.

We note that this does not affect our other parameters because we can choose the Reed-Solomon codes properly to correct a constant fraction of errors. For LPN security, the LPN instance is more secure when the noise rate is larger, and constant noise rate was referred to as *high noise* LPN in the literature [40].

5.3 Sub-Protocol For Secret Sharing Bits/Masks

Our final required subprotocol is one for generating secret shares of random bits, both to be used as masks for permuting the ciphertexts (see end of Section 3) and for LPN encryption (see Section 5.2).

We choose to work in a Galois Field of characteristic 2. Techniques used in the dishonest majority setting [38, 68] for sampling shares of random bits are not helpful here, since they require $O(n)$ communication (for sharing each bit) in the honest majority setting. Efficient honest majority techniques [13, 36] are known for generating secret shares of an unknown random value in the field. These techniques however, necessarily require the field from which random values are sampled to be linear in the number of parties. More recently, Cascudo et al. [27] proposed a way to extend these ideas for generating shares of uniform random binary values [7] embedded in a bigger field $\mathbb{F}$, with a similar efficiency.

We start by recalling the standard technique [13, 36] used for generating shares of random values in the field in *batches*, using a Vandermonde matrix of size $n \times (n-t)$. Specifically, each party secret shares a random value in the field, and then each party locally multiplies the shares that it receives from other parties with the Vandermonde matrix. Since every square sub-matrix of size $(n-t) \times (n-t)$ of a Vandermonde matrix is invertible and honest parties are expected to secret share truly random values, the result is that the parties obtain $O(n)$ secret shares of random, independent values. Overall, with $O(n^2)$ communication and computation, using the above approach, parties are able to generate $O(n)$ random sharings.

To generate shares of random bits, it is not sufficient to require the parties to simply start by secret sharing random bits instead of random values in $\mathbb{F}$. If the Vandermonde matrix contains elements in $\mathbb{F}$ (as is the case in initial works [13, 36]), even if the parties start with shares of bits, the shares obtained after multiplying input shares with this matrix will be of elements in $\mathbb{F}$ rather than that of bits. To address this issue, [27] observed that the generator matrix of any binary linear error correcting code (denoted by **binM**) is a *super-invertible matrix over* $\mathbb{F}_2$. The parties can now start by simply secret sharing random bits and when they multiply these shares with **binM**, the resulting shares will be of independent, random bits. This allows us to generate $O(n)$ random bit sharings with $O(n^2)$ communication and computation.

The same observation can also be used to generate *packed secret shares* of random bit-vectors. Each party simply sends packed secret sharing of vectors of random bits to the other parties. Each party then multiplies the received shares with the super-invertible bit matrix **binM**. This results in $O(n)$ packed secret sharings (containing n elements in each vector) with $O(n^2)$ communication and computation. A careful reader may have observed that this protocol yields shares of bits only if the parties originally start with sharings of bits (which cannot be guaranteed in the presence of malicious adversaries). As such, this protocol is only secure against a semi-honest adversary. We discuss malicious security for this protocol in Section 5.5.

5.4 An Appropriate $O(|\mathsf{C}|)$ MPC protocol

As discussed earlier, packed secret-sharing scheme (PSS) is a polynomial based linear secret sharing scheme that allows sharing a vector of secrets $\mathbf{v} = \{v_1, \ldots, v_\ell\}$, where $\ell \in O(n)$. Essentially, the dealer samples a random polynomial q of appropriate degree, such that for each $j \in [\ell]$, $q(\mathsf{slot}_j^{\mathsf{def}}) = v_j$ and each party P_i (for $i \in [n]$) gets a share $q(p_i)$ (where p_i is a publicly known field element that is unique to party P_i). Most existing $O(|\mathsf{C}|)$ MPC protocols use the same set of slots/points $\mathsf{slot}_1^{\mathsf{def}}, \ldots, \mathsf{slot}_\ell^{\mathsf{def}}$ in the polynomial for embedding secrets in all packed secret sharings used throughout the protocol. Using PSS, it is possible to evaluate a block of $O(n)$ gates at the same multiplicative depth in the circuit, in one shot.

Goyal et al.'s protocol [54] is a non-constant round, gate-by-gate evaluation style of protocol that slightly deviates from this approach. In this protocol, a unique slot/field element $\mathsf{slot}_g^{\mathsf{C}}$ is assigned for every gate g in the circuit and the following invariant is maintained throughout the protocol: let $g_1, \ldots, g_\ell$ be a block of gates that are evaluated simultaneously using PSS and let $\mathbf{z} = \{z_{g_1}, \ldots, z_{g_\ell}\}$ be output of these gates. Upon evaluating these gates, parties obtain a packed secret sharing of $\mathbf{z}$, where each z_{g_j} is embedded at the slot associated with gate g_j. Borrowing notion from [54], we use $[\mathbf{z}|\mathsf{pos}]$ to denote such a sharing, where $\mathsf{pos} = \{\mathsf{slot}_{g_1}^{\mathsf{C}}, \ldots, \mathsf{slot}_{g_\ell}^{\mathsf{C}}\}$.

Evaluating a Block of Gates. We now explain in more detail how a block of gates are evaluated in [54] using PSS. Let d denote the degree of the PSS. Let $g_1, \ldots, g_\ell$ be a block of multiplication or addition gates that we wish to evaluate and let $\mathbf{l} = \{l_{g_1}, \ldots, l_{g_\ell}\}$ be the set of left inputs to these gates. Further, let us assume that that for each $j \in [\ell]$, l_{g_j} was the output of some gate h_j. Given the

[7] More generally, Cascudo et al. [27] proposed an idea for generating shares of random values from any constant-sized field. In this work, we only focus on sampling from the Boolean field.

above invariant, this means that for each $j \in [\ell]$, there must exist some degree-d packed secret sharing of the form $[\mathbf{z}_j \,|\mathrm{pos}_j]_\mathrm{d}$, where $\mathbf{z}_j = \{\ldots, l_{g_j}, \ldots\}$ and $\mathrm{pos}_j = \{\ldots, \mathrm{slot}^\mathsf{C}_{h_j}, \ldots\}$. The next steps are as follows:

(1) *Bringing all left inputs to the same PSS:* In order to evaluate $g_1, \ldots, g_\ell$ simultaneously, the first step is to bring all left inputs $l_{g_1}, \ldots, l_{g_\ell}$ in the same PSS. This can be done by allowing the parties to locally multiply each $[\mathbf{z}_j \,|\mathrm{pos}_j]_\mathrm{d}$ with a degree-$(\ell-1)$ PSS of a unit vector $\mathbf{e}_j$ (i.e., where only the j-th term is 1 and all other terms are 0) of the form $[\mathbf{e}_j \,|\mathrm{pos}^h]_{\ell-1}$, where $\mathrm{pos}^h = \{\mathrm{slot}^\mathsf{C}_{h_1}, \ldots, \mathrm{slot}^\mathsf{C}_{h_\ell}\}$. The resulting degree-$(\mathrm{d}+\ell-1)$ sharing will be such that the value stored at position $\mathrm{slot}^\mathsf{C}_{h_j}$ is l_{g_j} and the values stored at other positions in pos^h are all 0. Adding all of these multiplied shares will result in shares of the form $[\mathbf{l} \,|\mathrm{pos}^h]_{\mathrm{d}+\ell-1} = \sum_{j\in[\ell]} [\mathbf{z}_j \,|\mathrm{pos}_j]_\mathrm{d} \cdot [\mathbf{e}_j \,|\mathrm{pos}^h]_{\ell-1}$.

(2) *Transforming to a PSS at default slots:* We now want to transform the above sharing $[\mathbf{l} \,|\mathrm{pos}^h]_{\mathrm{d}+\ell-1}$ into a sharing of the form $[\mathbf{l} \,|\mathrm{pos}_\mathrm{def}]_\mathrm{d}$, where $\mathrm{pos}_\mathrm{def} = \{\mathrm{slot}^\mathrm{def}_1, \ldots \mathrm{slot}^\mathrm{def}_\ell\}$ are some default slots used throughout the protocol that are independent from the ones associated the gates. We will discuss how this transformation is done shortly.

All the above steps are repeated for all the right input wire values $\mathbf{r} = \{r_{g_1}, \ldots, r_{g_l}\}$ to obtain a sharing of the form $[\mathbf{r} \,|\mathrm{pos}_\mathrm{def}]_\mathrm{d}$. If $g_1, \ldots, g_\ell$ were a block of addition gates, the parties can simply add their respective shares in $[\mathbf{l} \,|\mathrm{pos}_\mathrm{def}]_\mathrm{d}$ and $[\mathbf{r} \,|\mathrm{pos}_\mathrm{def}]_\mathrm{d}$ to obtain a sharing $[\mathbf{z} \,|\mathrm{pos}_\mathrm{def}]_\mathrm{d}$, where $\mathbf{z} = \{(l_{g_1} + r_{g_1}), \ldots, (l_{g_\ell} + r_{g_\ell})\}$. If $g_1, \ldots, g_\ell$ were a block of multiplication gates, the parties can use existing multiplication protocols [33] for computing packed shares $[\mathbf{z} \,|\mathrm{pos}_\mathrm{def}]_\mathrm{d}$ of the multiplied values, i.e., $\mathbf{z} = \{(l_{g_1} \cdot r_{g_1}), \ldots, (l_{g_\ell} \cdot r_{g_\ell})\}$.

Finally, in order to comply with the invariant, the last step in their protocol is to transform $[\mathbf{z} \,|\mathrm{pos}_\mathrm{def}]_\mathrm{d}$ into $[\mathbf{z} \,|\mathrm{pos}]_\mathrm{d}$, where $\mathrm{pos} = \{\mathrm{slot}^\mathsf{C}_{g_1}, \ldots, \mathrm{slot}^\mathsf{C}_{g_\ell}\}$ are the positions associated with gates $g_1, \ldots, g_\ell$. Next, we discuss how this transformation is done.

Share Transformation. Notice that in the above approach, we need to switch between sharings of the form $[\mathbf{x} \,|\mathrm{pos}_1]_{d_1}$ and $[\mathbf{x} \,|\mathrm{pos}_2]_{d_2}$. This can be done easily if the parties have access to secret sharings of random vectors of form $[\mathbf{r} \,|\mathrm{pos}_1]_{n-1}$ and $[\mathbf{r} \,|\mathrm{pos}_2]_{d_2}$. Indeed, given such sharings, the parties can do the following: (1) locally compute $[\mathbf{x}+\mathbf{r} \,|\mathrm{pos}_1]_{n-1}$, (2) reconstruct $\mathbf{x}+\mathbf{r}$, (3) compute $[\mathbf{x}+\mathbf{r} \,|\mathrm{pos}_2]_{d_2}$ and finally, (4) subtract the random sharing to get $[\mathbf{x} \,|\mathrm{pos}_2]_{d_2}$.

Prior approaches for generating such correlated random sharings $[\mathbf{r} \,|\mathrm{pos}_1]$ and $[\mathbf{r} \,|\mathrm{pos}_2]$ required $O(n^2)$ communication. Goyal et al. [54] propose a novel idea that enables efficient generation of such correlated randomness with $O(n)$ communication. Due to space constraints, we details details to the Appendix.

Key Generation and Garbling. For our multiparty garbling scheme, we also want to enable the parties to generate a secret sharing of random keys. In order to do this with $O(|\mathsf{C}|)$ total communication, we generate PSS of a random vector of keys. As discussed in Section 5.3, this can be done quite efficiently using known techniques [13, 36]. However, since shares of random values are generated in "batches" using this technique, when used for generating PSS, the secrets in the resulting packed shares are always stored at the same slots. While generating we ensure that these slots are always the default positions. This is also the case when we sample random sharings of bit masks and computing shares of the error vectors. When computing the ciphertext, we use the above share transformation protocol to move the above PSS of keys/masks/errors to another PSS where these values are all stored at the different slots associated with the gates/wires that they correspond to. We can now easily compute the garbling functionality using these values as input, and by relying on techniques from [54].

5.5 Adding Malicious Security

To ensure malicious security of our above approach, we need to thwart the following type of attacks:

- *Attack Type I:* The malicious parties can cause the ciphertexts to decrypt to an incorrect value by influencing error generation.
- *Attack Type II:* Any other potential attacks during the garbling phase (including at the time of key/mask generation), we need to ensure that the MPC protocol used for all the other computations in the garbling phase is also secure against malicious adversaries.

We first discuss how to handle the second type of attacks. Genkin et al. [47, 48] observed that most semi-honest, secret sharing (and packed secret sharing) based MPC protocols are also *private against malicious adversaries* until parties reconstruct the output shares. To add full-malicious security to such a protocol, the parties simply need to verify that the non-linear operations (i.e., non-scalar multiplications) in the circuit were honestly computed before reconstructing the output.

A recent line of works have showed how to incorporate these malicious security checks *efficiently* in the honest majority setting [31, 35, 38, 45]. The most popular kind of check is one where the parties sample a random sharing of a global MAC key (say kmac), which is essentially a random element in $\mathbb{F}$. Throughout the protocol, the parties perform every computation twice to maintain the following invariant: for every intermediate value z in the computation for which the parties hold a secret sharing (or packed secret sharing), they also hold a sharing of $(\mathrm{kmac} \cdot z)$. At the end of the parties compute a random linear combination of all the intermediate values and also compute a linear combination of all the MAC'ed intermediate values and essentially check whether the outcome of the second combination is kmac times that of the first combination. When working on a large field (i.e., exponential in the security parameter), it suffices to use a single MAC key. For smaller fields, the above check needs to be repeated for different MAC keys (to ensure negligible failure probability).

Goyal et al. [54] demonstrate how the above check seamlessly extends to their protocol and techniques. We rely on similar observations to ensure malicious security of most of our garbling protocol. Besides error generation, the only other sub-protocol that we use is the random bit sharing protocol for generating shares of masks. While this sub-protocol is already private against malicious adversaries (which follows from the observation of Genkin et al. [47, 48]), to ensure security of our garbling protocol, we also need to ensure it actually outputs valid shares of bits and not any other

field element. Indeed, if the adversary deviates from the protocol description, it could cause the parties to output (potentially invalid sharings) of any random field elements. The standard technique for checking if any given element b is 0 or 1, is to simply check if $b^2 \stackrel{?}{=} b$. We use the same idea. Upon receiving PSS of bits from the random bit sharing protocol, the parties multiply this sharing with itself (correctness of this multiplication checked using the above MAC based check) and at the end, we collective check if the above condition (i.e., $b^2 = b$) is met for all bits that were generated, in a single shot.

To counter the first type of attack (i.e., one that originates from incorrect error generation), we recall that the two main steps in our error generation sub-protocol are: (1) generating packed shares of random bits and (2) multiplying these bits. It is easy to see that security and correctness of both these steps can be ensured using the above ideas.

6 A SCALABLE MULTIPARTY GARBLING PROTOCOL

Having now described how we can instantiate all of the required components of our template, we can now proceed to describe the full protocol. Due to space constraints, we only provide an informal description of the semi-honest version of our protocol in Protocol 1 and defer the complete description of the malicious protocol and the required subprotocols to the full version.

For completeness, we provide a summary of the full protocol with malicious security below. Our protocol consists of three phases: circuit independent preprocessing, garbling, and reconstruction with evaluation.

In the circuit independent preprocessing phase, parties generate the randomness that is needed to permute, encrypt and create the garbled circuit. This mainly consists of enough packed secret sharings of wire keys and masks to cover all wires in the circuit which will be used to create the individual garbled tables. Because the LPN encryption scheme also requires additive errors from a particular distribution and are not dependent on any later computation, we also choose to generate these values in this phase. To achieve malicious security, a global mac key is also generated and is used to authenticate the shares in the keys and mask values on each wire. Additionally, because malicious parties may cause our randomness generation to output values that are not of the correct distribution, we retain some information for consistency checks that are needed on both randomness for LPN error generation and the randomly generated masks.

After key material has been generated, parties enter an online garbling phase. The first step of garbling is to take the packed secret shares and to move them out into positions that correspond to their positions associated with the gates in the circuit. This can be done using the share transformation algorithm of [54]. We now chunk up the gates in our circuit into packs of size ℓ. For each set of ℓ gates, we create new packed shares, packing together all input wire keys of the same type and value (where type is one of left,right, or out and wire value is 0 or 1), all output keys of the same value, and all output masks of the same type. The masks can now be used to select what output rows of the garbled tables will correspond to particular inputs and then in turn these selector bits can be used to select the output key for this row of the table. After obliviously selecting the selector bits and keys to encrypt, the LPN encryption scheme is used to encrypt the ciphertext using the packed secret shares of input keys and LPN errors generated in the circuit independent pre-processing step. We also generate packed shares for the input values, packing together input wire masks that belong to the same input client, as an optimization. Finally, throughout this whole phase, for every new value we compute, we also compute authentication tags. At the end of the garbling phase, we check that all these intermediate tags were computed correctly and finally run the consistency checks from the preprocessing phase. Note that at this point in time nothing sensitive has been computed, as we are just constructing the garbled circuit and no sensitive input from clients has been given.

In the last phase - reconstruction and evaluation - we start off by broadcasting input wire mask shares to the relevant parties. Clients add their input to the reconstructed mask, broadcast the results, and all parties can then compute a sharing of that client's input. This sharing of the client's input bit can then be used to also select for the wire key corresponding to the client's chosen value. All the garbled tables are reconstructed, along with the clients input and the output masks of the protocol (to ensure every party gets output). The rest of the protocol is a non-interactive evaluation of the garbled circuit.

7 PROTOCOL EVALUATION AND ANALYSIS

In this section, we attempt to get a better picture of the concrete performance of our protocol by analyzing its communication and computation costs. We first discuss some modifications to the protocol that improve its performance in practice followed by a discussion on the choice of optimal parameters for LPN and the binary super-invertible matrix. We then discuss the performance of our semi-honest and maliciously secure protocols and compare it to those of prior works. Our analysis will be centered around the performance of the pre-processing and garbling phases which constitute the communication intensive parts of our protocol.

7.1 Practical Protocol Optimizations

The following optimizations can help reduce the concrete costs of the protocol.

(1) **Pack circuit input wires separately.** Instead of packing keys and masks for blocks of all wires in the circuit together, separately pack the keys and masks for circuit input wires and the remaining wires (which correspond to output wires of individual gates) in the circuit. Then, $\mathbf{k}_b^{\text{out}}$ and λ_{out} need not be computed in step 2 as they are equal to the values sampled in the pre-processing phase.
(2) **Pack XOR and AND gates separately.** One way to choose gates to pack together in step 2 is to only pack the ciphertexts for garbled tables of the same gate type together, instead of packing by some arbitrary metric. While this does lead to slightly inefficient packing, it allows us to simplify parts of step 4. Otherwise, we would need to create packed shares of masks for each function gates could be computing and then for each gate select the mask from the relevant packed share.

Protocol 1: Semi-Honest Garbling Protocol

Pre-processing Phase:

(1) For $j \in [1, \frac{W}{\ell}]$ where W is the number of wires within the circuit, parties compute packed shares of a random mask $[\lambda]$, and packed sharings of keys $\{[\mathbf{k}_i^b]\}_{i\in[1,L_{\mathsf{lpn}}],b\in\{0,1\}}$

(2) For $j \in [1, \frac{G}{\ell}]$ where G is the number of gates, $k \in \{0, 1 \dots 3\}$, generate packed LPN errors $\{[\epsilon_i]\}_{i\in[1,Q_{\mathsf{lpn}}]}$

Garbling Phase:

(1) Transform packed shares of wires and keys from default positions to associated wire value positions pos as $\{[\mathbf{k}_i^0|\mathsf{pos}]\}_{i\in[1,L_{\mathsf{lpn}}]}, \{[\mathbf{k}_i^1|\mathsf{pos}]\}_{i\in[1,L_{\mathsf{lpn}}]}, [\lambda|\mathsf{pos}]$

(2) For $j \in [1, \frac{G}{\ell}]$, create new packed shares selecting all keys associated with the $\{\mathsf{left}, \mathsf{right}, \mathsf{out}\}$ input wires having value $\{0, 1\}$, and all wire masks denoted as $\{[\mathbf{k}_{b,i}^m|\mathsf{pos}]\}_{i\in[1,L_{\mathsf{lpn}}],b\in\{0,1\}}, [\lambda_m|\mathsf{pos}]$ for $m \in \{\mathsf{left}, \mathsf{right}, \mathsf{out}\}$

(3) Transform new packed shares to their default positions

(4) Select the plaintext to encrypt. For $\alpha, \beta \in \{0, 1\}$, compute the packed select bit determining output key for left wire value α and right wire value β according to gate function type. Let the result be $[\mathbf{s}^{\alpha,\beta}]$. For each $i \in [1, L_{\mathsf{lpn}}]$ securely compute $[\mathbf{k}_{\mathsf{active}}] = [\mathbf{s}_i^{\alpha,\beta}] \cdot ([\mathbf{k}_{1,i}^{\mathsf{out}}] - [\mathbf{k}_{0,i}^{\mathsf{out}}]) + [\mathbf{k}_{0,i}^{\mathsf{out}}]$ to obtain packed shares of keys to encrypt.

(5) Encode the plaintext and encrypt. Run an encoding procedure on a message containing $[\mathbf{k}_{\mathsf{active}}]$ concatenated with s. Run the LPN encryption algorithm using as the key for each $i \in [1, L_{\mathsf{lpn}}]$, $[\mathbf{k}_{\alpha,i}^{\mathsf{left}}] + [\mathbf{k}_{\beta,i}^{\mathsf{right}}]$ and errors $\{[\epsilon_i]\}_{i\in[1,Q_{\mathsf{lpn}}]}$

(6) For each input wire, pack together shares belonging to the same party and all keys associated with 0 and 1 values on these wires

Garbling Reconstruction and Evaluation

(1) For each client who will provide value on an input wire w, broadcast sharings in $[\lambda_w]$ to the party to allow reconstruction of λ_w. The client then broadcasts its masked input, from which can derive input shares of the mask and input keys

(2) For all packs of gates, for all rows in the circuit, reconstruct the ciphertexts $\{[\mathbf{c}_i]_{i\in[1,Q_{\mathsf{lpn}}]}\}$ and all output masks for the circuit.

(3) Evaluate the garbled circuit.

(3) **Reduce cost for computing mask bits.** For AND gates, we only need one multiplication to compute $[\mathbf{s}_i^{\alpha,\beta}]$ across all $\alpha, \beta \in \{0, 1\}$ for a given $i \in [1, L_{\mathsf{lpn}}]$. This reduces the cost of garbling an AND gate by 3 multiplications. See [84] for more details.

(4) **Replacing expensive protocols with degree reduction wherever possible.** Sometimes in the protocol, we need to perform a degree reduction on shares without changing their positions. For efficiency reasons, rather than use heavy weight sub-protocols that provide functionality which subsumes this, we generate random shares of degree t and $2t$ and perform a leader-based degree reduction similar to what is done in secure multiplication.

7.2 LPN Parameters

Our analysis of the security of LPN over larger fields follows that of Liu et al. [70]. The LPN parameters provide a trade-off between the security provided by the garbled circuit as well as the correctness error when evaluating the garbled circuit. Specifically, to correctly decrypt the ciphertext during evaluation, the weight of the noise vector $\mathbf{e}$, which follows the binomial distribution, should be lesser than half the distance of the error correcting code. Namely, $\Pr_{\mathbf{e}}[\mathsf{weight}(\mathbf{e}) \le \lfloor (d-1)/2 \rfloor] = \Pr[\mathsf{Binom}(Q, \tau) \le \lfloor (d-1)/2 \rfloor]$. On the other hand, a noise vector with very small weight would lower security.

For our protocols, we set the noise rate τ of our LPN-based encryption to be a constant, and require only a polynomial number of samples. We choose the parameters of a Reed-Solomon code to correct constant fraction of errors. To find the best parameters, we fix the noise rate τ_{lpn} and use binary search to find Q and V such that the distance $d = Q - V + 1$ of the Reed-Solomon code satisfies

$$\Pr[\mathsf{Binom}(Q, \tau) \le \lfloor (d-1)/2 \rfloor] \le 2^{-40},$$

while ensuring that the LPN parameters (Q, V, τ) provide 80-bits of security, as determined using the Python script provided by Liu et al. [70].

We find that for a correctness error of 2^{-40}, $Q = 555, V = 127, \tau = 2^{-2}$ are the optimal parameters for achieving 80-bit security and $Q = 785, V = 214, \tau = 2^{-2}$ are the optimal parameters for achieving 128-bit security. For the Reed-Solomon code we choose $[555, 128, 428]_q$ and $[785, 215, 571]_q$ respectively for 80-bit and 128-bit security.

7.3 Parameters for Binary Super-Invertible Matrices

We use a concatenation of an outer Reed Solomon code and an inner binary error-correcting code to obtain the binary super-invertible matrix.

In more detail, let the Reed Solomon code parameters be $[Q_r, L_r, d_r]_q$, where $d_r = Q_r - L_r + 1$ and $q \ge Q_r$ is a power of 2. Let the inner code parameters be $[Q_i, L_i, d_i]_2$ with $q = 2^{L_i}$. Then by Theorem 2.2, the concatenated code has parameters $[Q_rQ_i, L_rL_i, d_rd_i]_2$. Hence, we need $Q_rQ_i \ge Q$. However, when $Q_rQ_i > Q$, then we need to truncate $(Q_rQ_i - Q)$-rows of the generating matrix. This causes a loss of $(Q_rQ_i - Q)$ in the distance, and we thus obtain a $[Q, L_rL_i, d_rd_i - (Q_rQ_i - Q)]_2$-code. By Theorem 2.1, if we have $\lfloor Q/3 \rfloor$ malicious parties, then we need $d_rd_i - (Q_rQ_i - Q) + 1 \ge \lfloor Q/3 \rfloor$. In summary, we need to choose the parameters which maximize message length L_rL_i with the following constraints.

$$\begin{aligned} Q_r \cdot Q_i &\ge Q \\ q = 2^{L_i} &\ge Q_r \\ Q_r \cdot Q_i - d_r \cdot d_i &\le Q - \lfloor Q/3 \rfloor. \end{aligned}$$

In our setting, we assume $Q = n$, i.e., the number of parties. If we use Reed Solomon Codes and the inner code with constant rate, then the resulting concetenation code will also have constant rate. In this case, it is easy to see that $L_rL_i \in O(n)$. We can now use the generator matrix of $[Q, L_rL_i, d_rd_i - (Q_rQ_i - Q)]_2$-code as our binary super-invertible matrix. The dimension of this matrix will be $Q \times L_rL_i$, i.e., $n \times O(n)$, which is what we want.

For concrete parameters, we take the BCH codes [23] as the inner codes, and use a Python script to enumerate all combinations of the Reed Solomon codes and the BCH codes to find the largest

Circuit	n	t	ℓ	Pre-Processing Runtime (s)	Pre-Processing Comm. (MB)	Pre-Processing Size (MB)	Garbling Runtime (s)	Garbling Comm. (MB)
AES-128	128	31	33	128.413	253.200	40.950	13.411	26.587
	256	63	65	95.933	107.270	21.212	10.262	13.749
	512	127	129	110.776	58.742	12.543	15.527	8.094
SHA-256	128	31	33	-	-	152.536	46.167	99.028
	256	63	65	453.787	402.479	79.155	39.271	51.294
	512	127	129	441.084	213.369	41.888	40.797	27.074

Table 1: Runtime and per party communication cost of our implementation of the semi-honest variant of our protocol when each party is run with 2 threads. n is the number of parties, $t = \lfloor (n-1)/4 \rfloor$ is the corruption threshold, and ℓ is the packing parameter. The security parameters are set to $\kappa_s = 40$ and $\kappa_c = 80$. AES-128 has 36663 gates and SHA-256 has 114107 gates.

possible L_r. Our script also enumerates random linear codes achieving Gilbert–Varshamov bound as the inner code. Here we list some concrete parameters. For $n = 256$ and $t = 63$, we choose $[16, 6, 11]_{2^7}$-Reed Solomon code concatenated with $[16, 7, 6]_2$-BCH code. For $n = 512$ and $t = 127$, we choose $[32, 11, 22]_{2^7}$-Reed Solomon code concatenated with $[16, 7, 6]_2$-BCH code.

7.4 Evaluation of our Semi-Honest Secure Protocol

To evaluate the concrete performance of our protocol, we implement the semi-honest variant in Rust [8] and benchmark its performance in realistic deployment scenarios, executing common circuits with hundreds of parties located in different regions of the US. To do this we make use of publicly available cloud services provided by AWS. Our network set-up consists of a number of `c4.large` instances spread across the following AWS regions: `us-east-1`, `us-east-2`, and `us-west-2`. A `c4.large` is equipped with Intel(R) Xeon(R) E5-2666 processor and consists of 2 vCPUs and 3.75 GB of RAM. We used the MATRIX library [9] to orchestrate experiments over AWS. Our implementation is multi-threaded and makes use of asynchronous I/O to run protocols concurrently. We use the Fast Galois Field Arithmetic Library [79] for finite field arithmetic in our implementation and use the circuit descriptions available at [5] for our experiments. We run each experiment 5 times and report the average.

Table 1 summarizes the runtime and communication cost of the pre-processing and garbling phases as well as the size of the pre-processing material output by each party at the end of the pre-processing phase when garbling the AES-128 and SHA-256 circuits with 128, 256, and 512 parties whilst tolerating $t = \lfloor (n-1)/4 \rfloor$ corruptions. Our benchmarks indicate that our protocol is practical and can scale to a large number of parties since the runtime does not vary significantly with the number of parties and depends mainly on the size of the circuit being evaluated. The pre-processing phase for AES-128 takes a maximum of 128.4s and for SHA-256 takes a maximum of 453.8s. The garbling phase takes at most 15.5s for AES-128 and at most 46.1s for SHA-256. As expected, the per-party communication cost as well as the size of the pre-processing material decreases as the number of parties increases, owing to the $O(|C|)$ communication complexity. Our implementation's memory consumption exceeded the `c4.large` instance's 3.75 GB limit when running the pre-processing phase for SHA-256 with 128 parties. We note that such overheads in memory can be avoided by generating the pre-processing material in smaller batches instead of computing it all at once in the minimum number of rounds.

Table 2 compares the performance of the protocol with different corruption thresholds, when garbling AES-128 with 256 parties. We observe that both the runtime and communication costs decrease with the corruption threshold. Specifically, when tolerating $\frac{1}{6}$-th corruption instead of $\frac{1}{3}$-rd corruption, we notice a 2.7× improvement in the runtime for pre-processing and a 1.8× improvement in the runtime for the garbling phase. A minor irregularity is observed in the runtime of the garbling phase where it increases by 0.73s when tolerating $\frac{1}{6}$-th corruption compared to when tolerating $\frac{1}{5}$-th corruption. Note that a smaller corruption threshold t implies a larger packing parameter ℓ which in turn implies computing over fewer secret shares to garble the same circuit. However, a larger packing parameter also requires sharing secrets over a larger degree polynomial and increases the computation required per share. While the net effect implies constant computation complexity, the observed irregularity might be an artifact of the implementation due to the discussed effects of a larger packing parameter.

7.4.1 Comparison to Prior Work. Ben-Efraim and Omri [19] present efficient multiparty garbling protocols in the honest majority setting. We restrict our discussion to their semi-honest secure protocols since they do not instantiate the pre-processing phase for their maliciously secure protocols and provide benchmarks only for the former. They present two semi-honest protocols: BGW3_{opt} that can tolerate up to $t < \frac{n}{2}$ corruptions and the more efficient BGW2_{opt} protocol that is secure up to $t < \frac{n}{3}$ corruptions, both of which have quadratic computation and communication complexity in the number of parties n. We compare the performance of our semi-honest protocol when run with $t = \frac{(n-1)}{3}$ to the performance of BGW2_{opt}. BGW2_{opt} has a total runtime of 0.109s when garbling AES-128 with 13 parties over LAN. Scaling the runtime, given the protocol's quadratic growth in computation and communication costs with the number of parties, suggests that the protocol would take at least 42.27s to garble AES-128 with 256 parties over LAN. In comparison, from Table 2, our protocol takes a total of 218.67s to garble AES-128 with 256 parties. Thus, our semi-honest protocol has comparable performance despite being run over a network with

[8] github.com/adishegde/scalable_garbling

t	ℓ	Pre-Processing Runtime (s)	Pre-Processing Comm. (MB)	Pre-Processing Size (MB)	Garbling Runtime (s)	Garbling Comm. (MB)
$85 = \lfloor (n-1)/3 \rfloor$	43	200.904	326.251	45.568	17.777	28.530
$63 = \lfloor (n-1)/4 \rfloor$	65	95.933	107.270	21.212	10.262	13.749
$51 = \lfloor (n-1)/5 \rfloor$	77	75.935	77.011	16.486	8.964	10.841
$42 = \lfloor (n-1)/6 \rfloor$	86	72.441	62.195	13.456	9.696	8.955

Table 2: Comparison of the runtime and per party communication of the semi-honest variant of our protocol with different corruption thresholds when garbling the AES-128 circuit with $n = 256$ parites where each party is run with 2 threads. t is the corruption threshold, and ℓ is the packing parameter. The security parameters are set to $\kappa_s = 40$ and $\kappa_c = 80$.

lower bandwidth and higher latency. Moreover, BGW2_{opt} takes 34.17s to garble SHA-256 with 31 parties over LAN which would imply a runtime of at least 2330.38s when garbling SHA-256 with 256 parties over LAN. On the other hand, scaling the runtime of our protocol from Table 2, we expect our protocol to take 680.61s when garbling SHA-256 with 256 parties whilst tolerating $\frac{1}{3}$-rd corruption. This indicates that for larger circuits, our protocol outperforms BGW2_{opt} despite being run over a slower network while for smaller circuits we expect our protocol to have similar or better runtimes when run over identical network conditions.

7.5 Evaluation Of Maliciously Secure Protocol

While we do not implement our maliciously secure protocol, we evaluate its performance by estimating its communication and computation costs. To estimate communication costs, we wrote a `python` script that outputs the communication required for each phase of the protocol by computing the number of bits communicated by all parties in every sub-protocol. To estimate the concrete computation costs of our protocol, we first benchmarked the time required for individual field operations (addition and multiplication) followed by programmatically estimating the total number of field operations carried out in a protocol execution using a `python` script. As in our implementation of the semi-honest protocol, we used the Fast Galois Field Arithmetic Library [79] for field arithmetic. We found that on a `c4.large` instance (cf. Section 7.4), a field multiplication takes an average time of 5.6319e-10s and a field multiplication takes on average 1.0079e-8s. For the sake reproducibility, the scripts used for estimating the communication and computation costs as well as benchmarking the time for field operations have been included in the associated github repository[9].

Table 3 summarizes the estimated computation and communication costs for garbling AES-128 and SHA-256 with 128, 256, and 512 parties whilst tolerating $t = \lfloor (n-1)/4 \rfloor$ corruptions. As expected, the per party communication cost decreases significantly with an increase in the number of parties. The communication required for the maliciously secure pre-processing and garbling phases is around 5.05× and 5.89× the communication required for the semi-honest secure pre-processing and garbling phases respectively. To better understand how the computational overhead affects the total runtime, we estimated the computation time for the semi-honest protocol too and found that the runtime of our implementation (cf. Section 7.4) was on average 3.14× the estimated computation time for the pre-processing phase and 2.54× the estimated computation time for the garbling phase. It is reasonable to expect that the relationship between the estimated computation time and total runtime would be similar for the malicious protocol. Thus, the maliciously secure protocol is expected to have reasonable runtime in practice, and as in the case of the semi-honest protocol the runtime is not expected to vary significantly with the number of parties but depend mainly on the size of the circuit being evaluated.

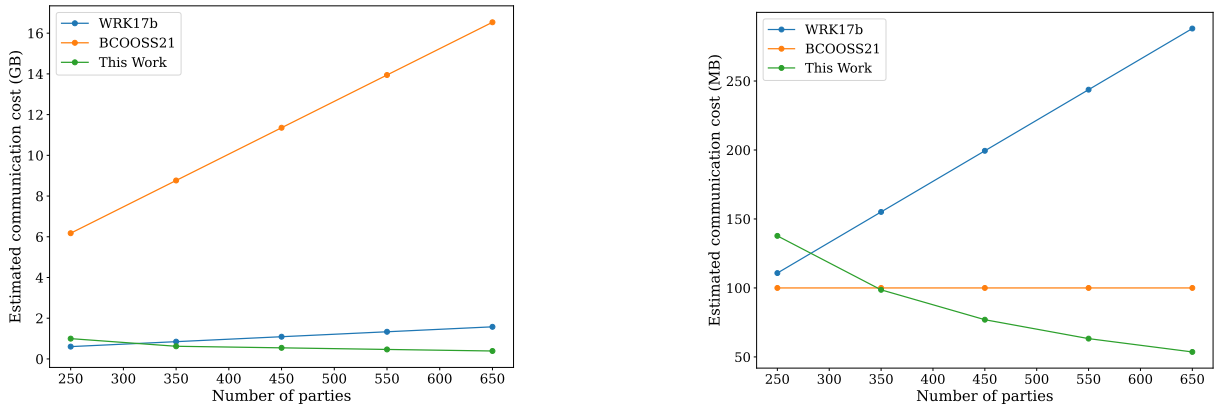
7.5.1 Comparison To Prior Works. Ben-Efraim et al. [16] construct a BMR-style protocol in the dishonest majority setting which only requires $O(n)$ communication per party in the garbling phase and makes use of a somewhat similar LPN-based encryption scheme. We also compare our protocol against the authenticated garbling protocol of Wang et al. [85] which we denote by WRK17b. While WRK17b and the protocol of [16] can tolerate at most $t = n - 1$ corruptions when run with n parties, Ben-Efraim et al. [16] propose an efficient variant when tolerating a sub-optimal corruption threshold. Specifically, assuming the presence of n/c honest parties, where $1 < c < n$, allows for a more communication efficient protocol especially when $n/c > \kappa_s$. As done in the performance evaluation of [16], we set $c = 5$ for the purpose of our analysis and refer to this protocol as BCOOSS21. Since the protocols we compare tolerate a different corruption threshold, we consider the case when all protocols are run with the same number of parties as well as when each protocol is run with a different number of parties but tolerates the same number of corruptions. We set the corruption threshold to $t = \lfloor (n-1)/4 \rfloor$ for our protocol in all cases. Wherever required, we extrapolate the benchmarks reported in [85] and [16] to estimate the communication cost of the protocols when run with a larger number of parties. We use linear interpolation for this extrapolation since the per party communication cost of WRK17b in the pre-processing and garbling phases and BCOOSS21 in the pre-processing phase, grows linearly with the total number of parties.

Figure 1 summarizes the per party communication cost of the protocols when each protocol is run with the same number of parties to garble AES-128. In the pre-processing phase, the communication cost of WRK17b and BCOOSS21 is around 0.61× and 6.19× the communication cost of our protocol respectively when $n = 250$ and increases to around 4.04× and 42.39× the communication cost of our protocol when $n = 650$. In the garbling phase, the communication cost of WRK17b and BCOOSS21 is around 0.80× and 0.73× the communication cost of our protocol when $n = 250$ and increases to around 5.37× and 1.86× the communication cost

[9] github.com/adishegde/scalable_garbling

Circuit	n	t	ℓ	Pre-Processing		Garbling	
				Comp. Time (s)	Comm. (MB)	Comp. Time (s)	Comm. (MB)
AES-128	128	31	33	≈ 200	≈ 1168	≈ 21	≈ 163
	256	63	65	≈ 199	≈ 573	≈ 19	≈ 83
	512	127	129	≈ 202	≈ 292	≈ 18	≈ 42
SHA-256	128	31	33	≈ 737	≈ 4301	≈ 79	≈ 602
	256	63	65	≈ 735	≈ 2111	≈ 71	≈ 308
	512	127	129	≈ 744	≈ 1075	≈ 67	≈ 155

Table 3: Estimated computation time and per party communication cost of the maliciously secure protocol when each party is run with 2 threads. n is the number of parties, $t = \lfloor (n-1)/4 \rfloor$ is the corruption threshold, and ℓ is the packing parameter. The security parameters are set to $\kappa_s = 40$ and $\kappa_c = 80$. AES-128 has 36663 gates and SHA-256 has 114107 gates.



(a) Pre-processing phase **(b) Garbling phase**

Figure 1: Comparison of estimated per party communication cost when garbling AES-128 with different multiparty garbling protocols, where each protocol is run with the same number of parties. We set the corruption threshold to $t = \lfloor \frac{(n-1)}{4} \rfloor$ for our protocol. The security parameters are set to $\kappa_s = 40$ and $\kappa_c = 128$ for all protocols.

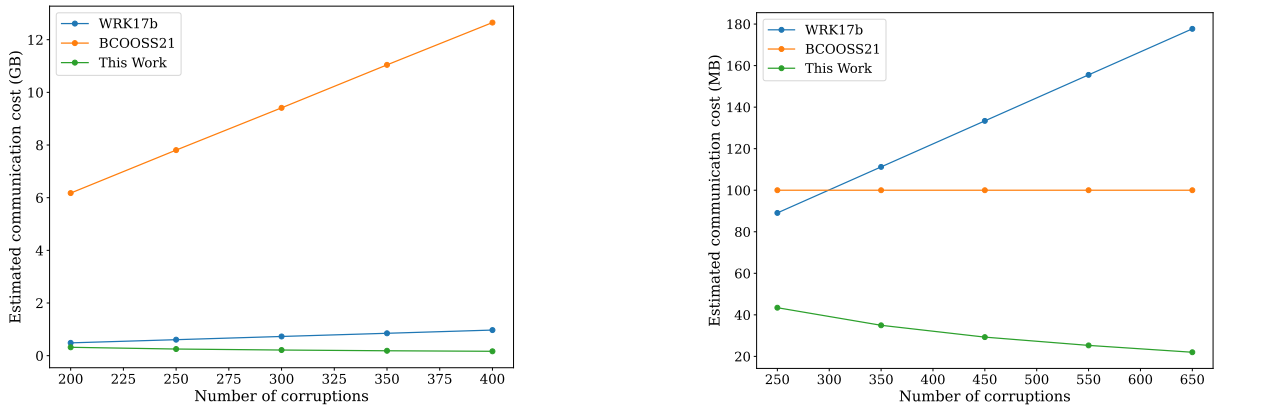
of our protocol when $n = 650$. Thus, the overall communication costs of our protocol, across both phases, is lower than that of WRK17b starting at around 350 parties while it is lower than that of BCOOSS21 even with 250 parties.

Figure 2 summarizes the per party communication cost of the protocols when each protocol tolerates the same number of corruptions when garbling AES-128. In this case, our protocol is run with approximately 4× the number of parties as in WRK17b and 3.2× the number of parties as in BCOOSS21 to ensure that all protocols tolerate the same number of corruptions. The presence of a large number of parties, leads to significantly lower communication overhead for our protocols compared to that of WRK17b and BCOOSS21. In the pre-processing phase, the per party communication cost of our protocol is 1.53× and 19.44× lower than that of WRK17b and BCOOSS21 respectively when $t = 200$ and up to 5.84× and 76.02× lower when $t = 400$. In the garbling phase, the per party communication cost of WRK17b and BCOOSS21 is around 2.05× and 2.3× the per party communication cost of our protocol respectively when $t = 200$, and up to 8.08× and 4.55× the per party communication cost of our protocol when $t = 400$. Moreover, as discussed previously, we do not expect the runtime of our protocols to change significantly with the number of parties and so we expect our protocol to outperform WRK17b and BCOOSS21 in these settings.

8 RELATED WORK

There is a long history of pushing towards $O(|C|)$ MPC [33, 34, 47], that has recently resulted in *linear round* (ie. communication rounds linear in the depth of the circuit) MPC protocols with $O(|C|)$ communication complexity that are concretely efficient [12, 52–54]. Our work builds on techniques proposed in these works, but applies them to the constant-round setting. Most relevant to our protocol, we use the share transformation protocol proposed by Goyal et al. [54] (see Section 5.4) as a subprotocol in order to achieve our result.

There are two popular templates for achieving constant round MPC. The first relies on multiparty variants of fully homomorphic encryption [7, 26, 51, 74, 75]. While improving the efficiency of FHE is an active area of research, this approach currently remains very far from practical. The second template, first proposed by Beaver, Micali and Rogaway (BMR) [11] relies on the observation that garbling a circuit [86] can be performed in constant depth.



(a) Pre-processing phase **(b) Garbling phase**

Figure 2: Comparison of estimated per party communication cost when garbling AES-128 with different multiparty garbling protocols, where each protocol is run to tolerate the same number of corruptions. We set the number of parties to be $n = 4t + 1$ for our protocol. The security parameters are set to $\kappa_s = 40$ and $\kappa_c = 128$ for all protocols.

In our work, we focus on this second approach: the problem of multiparty garbling.

The BMR approach has been the subject of significant research and has recently lead to asymptotically efficient constructions with garbled circuit specifications that can be evaluated quickly in practice [10, 16–18, 32, 46, 58–61, 68, 69, 76, 85]. While the original approach required non-black-box use of cryptography, Damgård and Ishai [32] proposed a black-box technique for multi party garbling, paving the way towards more efficient constructions.

Ben-Efraim, Lindell, and Omri [18] showed how to leverage LWE to garble a circuit with an evaluation complexity of $O(n|C|)$ per-party, improving on the prior $O(n^2|C|)$ per-party complexity of Lindell et al. [68]. Ben-Efraim et al. [16] then further optimized the output evaluation phase to require only $O(|C|)$ per-party local computation (after reconstruction) using an LPN based encryption scheme. Additionally, their protocol features a online garbling phase with total communication complexity $O(n|C|)$, but their circuit independent preprocessing phase still has total communication complexity $O(n^2|C|)$. They achieve this result by reducing the size of the garbled tables to be constant in the number of parties. Their scheme uses an LPN-based encryption scheme that is both key-homomorphic and message-homomorphic, further demonstrating linearly homomorphic cryptographic primitives can produce concretely efficient protocols [21, 29, 30, 38, 65, 78].

Finally we note that it is possible to garble arithmetic functionalities [4, 8], at a high cost. Ben-Ephraim et al. [15] and Makri et al. [71] study the feasibility of computing such function within a MPC protocol.

ACKNOWLEDGMENTS

Gabriel Kaptchuk is supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project and is supported by DARPA under Agreement No. HR00112020021. Gabrielle Beck and Aditya Hegde were supported by DARPA under Contract No. HR001120C0084. Aarushi Goel, Aditya Hegde, Abhishek Jain and Zhengzhong Jin were supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. Abhishek Jain was additionally supported in part by JP Morgan Faculty Award, and research gifts from Ethereum, Stellar and Cisco. Zhengzhong Jin was additionally supported in part by DARPA under Agreement No. HR00112020023 and by an NSF grant CNS-2154149. This work was done in part when Aarushi Goel and Zhengzhong Jin were students at Johns Hopkins University. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

REFERENCES

[1] Michael Alekhnovich. 2003. More on Average Case vs Approximation Complexity. In *44th FOCS*. IEEE Computer Society Press, 298–307. https://doi.org/10.1109/SFCS.2003.1238204

[2] Benny Applebaum, Jonathan Avron, and Christina Brzuska. 2015. Arithmetic Cryptography: Extended Abstract. In *ITCS 2015*, Tim Roughgarden (Ed.). ACM, 143–151. https://doi.org/10.1145/2688073.2688114

[3] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. 2017. Secure Arithmetic Computation with Constant Computational Overhead. In *CRYPTO 2017, Part I (LNCS, Vol. 10401)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Heidelberg, 223–254. https://doi.org/10.1007/978-3-319-63688-7_8

[4] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. 2011. How to Garble Arithmetic Circuits. In *52nd FOCS*, Rafail Ostrovsky (Ed.). IEEE Computer Society Press, 120–129. https://doi.org/10.1109/FOCS.2011.40

[5] David Archer, Victor Arribas Abril, Steve Lu, Pieter Maene, Nele Mertens, Danilo Sijacic, and Nigel Smart. [n. d.]. 'Bristol Fashion' MPC Circuits. https://homes.esat.kuleuven.be/~nsmart/MPC/

[6] David Archer, Amy O'Hara, Rawane Issa, and Stephanie Straus. 2021. Sharing Sensitive Department of Education Data Across Organizational Boundaries Using Secure Multiparty Computation.

[7] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Heidelberg, 483–501. https://doi.org/10.1007/978-3-642-29011-4_29

[8] Marshall Ball, Tal Malkin, and Mike Rosulek. 2016. Garbling Gadgets for Boolean and Arithmetic Circuits. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 565–577. https://doi.org/10.1145/2976749.2978410

[9] Assi Barak, Martin Hirt, Lior Koskas, and Yehuda Lindell. 2018. An End-to-End System for Large Scale P2P MPC-as-a-Service and Low-Bandwidth MPC for

Weak Participants. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 695–712. https://doi.org/10.1145/3243734.3243801

[10] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. 2020. Efficient Constant-Round MPC with Identifiable Abort and Public Verifiability. In *CRYPTO 2020, Part II (LNCS, Vol. 12171)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 562–592. https://doi.org/10.1007/978-3-030-56880-1_20

[11] Donald Beaver, Silvio Micali, and Phillip Rogaway. 1990. The Round Complexity of Secure Protocols (Extended Abstract). In *22nd ACM STOC*. ACM Press, 503–513. https://doi.org/10.1145/100216.100287

[12] Gabrielle Beck, Aarushi Goel, Abhishek Jain, and Gabriel Kaptchuk. 2021. Order-C Secure Multiparty Computation for Highly Repetitive Circuits. In *EUROCRYPT 2021, Part II (LNCS, Vol. 12697)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, Heidelberg, 663–693. https://doi.org/10.1007/978-3-030-77886-6_23

[13] Zuzana Beerliová-Trubíniová and Martin Hirt. 2008. Perfectly-Secure MPC with Linear Communication Complexity. In *TCC 2008 (LNCS, Vol. 4948)*, Ran Canetti (Ed.). Springer, Heidelberg, 213–230. https://doi.org/10.1007/978-3-540-78524-8_13

[14] Assaf Ben-David, Noam Nisan, and Benny Pinkas. 2008. FairplayMP: a system for secure multi-party computation. In *ACM CCS 2008*, Peng Ning, Paul F. Syverson, and Somesh Jha (Eds.). ACM Press, 257–266. https://doi.org/10.1145/1455770.1455804

[15] Aner Ben-Efraim. 2018. On Multiparty Garbling of Arithmetic Circuits. In *ASIACRYPT 2018, Part III (LNCS, Vol. 11274)*, Thomas Peyrin and Steven Galbraith (Eds.). Springer, Heidelberg, 3–33. https://doi.org/10.1007/978-3-030-03332-3_1

[16] Aner Ben-Efraim, Kelong Cong, Eran Omri, Emmanuela Orsini, Nigel P. Smart, and Eduardo Soria-Vazquez. 2021. Large Scale, Actively Secure Computation from LPN and Free-XOR Garbled Circuits. In *EUROCRYPT 2021, Part III (LNCS, Vol. 12698)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, Heidelberg, 33–63. https://doi.org/10.1007/978-3-030-77883-5_2

[17] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. 2016. Optimizing Semi-Honest Secure Multiparty Computation for the Internet. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 578–590. https://doi.org/10.1145/2976749.2978347

[18] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. 2017. Efficient Scalable Constant-Round MPC via Garbled Circuits. In *ASIACRYPT 2017, Part II (LNCS, Vol. 10625)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Heidelberg, 471–498. https://doi.org/10.1007/978-3-319-70697-9_17

[19] Aner Ben-Efraim and Eran Omri. 2019. Concrete Efficiency Improvements for Multiparty Garbling with an Honest Majority. In *LATINCRYPT 2017 (LNCS, Vol. 11368)*, Tanja Lange and Orr Dunkelman (Eds.). Springer, Heidelberg, 289–308. https://doi.org/10.1007/978-3-030-25283-0_16

[20] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *20th ACM STOC*. ACM Press, 1–10. https://doi.org/10.1145/62212.62213

[21] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT 2011 (LNCS, Vol. 6632)*, Kenneth G. Paterson (Ed.). Springer, Heidelberg, 169–188. https://doi.org/10.1007/978-3-642-20465-4_11

[22] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. 2013. Key Homomorphic PRFs and Their Applications. In *CRYPTO 2013, Part I (LNCS, Vol. 8042)*, Ran Canetti and Juan A. Garay (Eds.). Springer, Heidelberg, 410–428. https://doi.org/10.1007/978-3-642-40041-4_23

[23] R.C. Bose and D.K. Ray-Chaudhuri. 1960. On a class of error correcting binary group codes. *Information and Control* 3, 1 (1960), 68–79. https://doi.org/10.1016/S0019-9958(60)90287-4

[24] Sean Bowe, Ariel Gabizon, and Matthew D. Green. 2019. A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK. In *FC 2018 Workshops (LNCS, Vol. 10958)*, Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala (Eds.). Springer, Heidelberg, 64–77. https://doi.org/10.1007/978-3-662-58820-8_5

[25] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing Vector OLE. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 896–912. https://doi.org/10.1145/3243734.3243868

[26] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. 2017. Four Round Secure Computation Without Setup. In *TCC 2017, Part I (LNCS, Vol. 10677)*, Yael Kalai and Leonid Reyzin (Eds.). Springer, Heidelberg, 645–677. https://doi.org/10.1007/978-3-319-70500-2_22

[27] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. 2018. Amortized Complexity of Information-Theoretically Secure MPC Revisited. In *CRYPTO 2018, Part III (LNCS, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 395–426. https://doi.org/10.1007/978-3-319-96878-0_14

[28] David Chaum, Claude Crépeau, and Ivan Damgård. 1988. Multiparty Unconditionally Secure Protocols (Abstract) (Informal Contribution). In *CRYPTO'87 (LNCS, Vol. 293)*, Carl Pomerance (Ed.). Springer, Heidelberg, 462. https://doi.org/10.1007/3-540-48184-2_43

[29] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. 2020. Multiparty Generation of an RSA Modulus. In *CRYPTO 2020, Part III (LNCS, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 64–93. https://doi.org/10.1007/978-3-030-56877-1_3

[30] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. 2020. Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. Cryptology ePrint Archive, Report 2020/374. https://eprint.iacr.org/2020/374.

[31] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. 2018. Fast Large-Scale Honest-Majority MPC for Malicious Adversaries. In *CRYPTO 2018, Part III (LNCS, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 34–64. https://doi.org/10.1007/978-3-319-96878-0_2

[32] Ivan Damgård and Yuval Ishai. 2005. Constant-Round Multiparty Computation Using a Black-Box Pseudorandom Generator. In *CRYPTO 2005 (LNCS, Vol. 3621)*, Victor Shoup (Ed.). Springer, Heidelberg, 378–394. https://doi.org/10.1007/11535218_23

[33] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. 2010. Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography. In *EUROCRYPT 2010 (LNCS, Vol. 6110)*, Henri Gilbert (Ed.). Springer, Heidelberg, 445–465. https://doi.org/10.1007/978-3-642-13190-5_23

[34] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. 2008. Scalable Multiparty Computation with Nearly Optimal Work and Resilience. In *CRYPTO 2008 (LNCS, Vol. 5157)*, David Wagner (Ed.). Springer, Heidelberg, 241–261. https://doi.org/10.1007/978-3-540-85174-5_14

[35] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In *ESORICS 2013 (LNCS, Vol. 8134)*, Jason Crampton, Sushil Jajodia, and Keith Mayes (Eds.). Springer, Heidelberg, 1–18. https://doi.org/10.1007/978-3-642-40203-6_1

[36] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and Unconditionally Secure Multiparty Computation. In *CRYPTO 2007 (LNCS, Vol. 4622)*, Alfred Menezes (Ed.). Springer, Heidelberg, 572–590. https://doi.org/10.1007/978-3-540-74143-5_32

[37] Ivan Damgård, Claudio Orlandi, and Mark Simkin. 2018. Yet Another Compiler for Active Security or: Efficient MPC Over Arbitrary Rings. In *CRYPTO 2018, Part II (LNCS, Vol. 10992)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 799–829. https://doi.org/10.1007/978-3-319-96881-0_27

[38] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Heidelberg, 643–662. https://doi.org/10.1007/978-3-642-32009-5_38

[39] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS 2015*. The Internet Society.

[40] Nico Döttling. 2015. Low Noise LPN: KDM Secure Public Key Encryption and Sample Amplification. In *PKC 2015 (LNCS, Vol. 9020)*, Jonathan Katz (Ed.). Springer, Heidelberg, 604–626. https://doi.org/10.1007/978-3-662-46447-2_27

[41] Nico Döttling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto Trifiletti. 2017. TinyOLE: Efficient Actively Secure Two-Party Computation from Oblivious Linear Function Evaluation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 2263–2276. https://doi.org/10.1145/3133956.3134024

[42] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. 2022. TurboPack: Honest Majority MPC with Constant Online Communication. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 951–964. https://doi.org/10.1145/3548606.3560633

[43] Andre Esser, Robert Kübler, and Alexander May. 2017. LPN Decoded. In *CRYPTO 2017, Part II (LNCS, Vol. 10402)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Heidelberg, 486–514. https://doi.org/10.1007/978-3-319-63715-0_17

[44] Matthew K. Franklin and Moti Yung. 1992. Communication Complexity of Secure Computation (Extended Abstract). In *24th ACM STOC*. ACM Press, 699–710. https://doi.org/10.1145/129712.129780

[45] Jun Furukawa and Yehuda Lindell. 2019. Two-Thirds Honest-Majority MPC for Malicious Adversaries at Almost the Cost of Semi-Honest. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 1557–1571. https://doi.org/10.1145/3319535.3339811

[46] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. 2012. Concurrently Secure Computation in Constant Rounds. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Heidelberg, 99–116. https://doi.org/10.1007/978-3-642-29011-4_8

[47] Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. 2015. Efficient Multiparty Computation: From Passive to Active Security via Secure SIMD Circuits. In *CRYPTO 2015, Part II (LNCS, Vol. 9216)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.). Springer, Heidelberg, 721–741. https://doi.org/10.1007/978-3-

662-48000-7_35
[48] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. 2014. Circuits resilient to additive attacks with applications to secure computation. In *46th ACM STOC*, David B. Shmoys (Ed.). ACM Press, 495–504. https://doi.org/10.1145/2591796.2591861
[49] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, Michael Mitzenmacher (Ed.). ACM Press, 169–178. https://doi.org/10.1145/1536414.1536440
[50] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229. https://doi.org/10.1145/28395.28420
[51] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. 2015. Constant-Round MPC with Fairness and Guarantee of Output Delivery. In *CRYPTO 2015, Part II (LNCS, Vol. 9216)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.). Springer, Heidelberg, 63–82. https://doi.org/10.1007/978-3-662-48000-7_4
[52] S. Dov Gordon, Daniel Starin, and Arkady Yerukhimovich. 2021. The More the Merrier: Reducing the Cost of Large Scale MPC. In *EUROCRYPT 2021, Part II (LNCS, Vol. 12697)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, Heidelberg, 694–723. https://doi.org/10.1007/978-3-030-77886-6_24
[53] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. 2021. Unconditional Communication-Efficient MPC via Hall's Marriage Theorem. In *CRYPTO 2021, Part II (LNCS, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Virtual Event, 275–304. https://doi.org/10.1007/978-3-030-84245-1_10
[54] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. 2022. Sharing Transformation and Dishonest Majority MPC with Packed Secret Sharing. In *CRYPTO 2022, Part IV (LNCS, Vol. 13510)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Heidelberg, 3–32. https://doi.org/10.1007/978-3-031-15985-5_1
[55] Vipul Goyal and Yifan Song. 2020. Malicious Security Comes Free in Honest-Majority MPC. Cryptology ePrint Archive, Report 2020/134. https://eprint.iacr.org/2020/134.
[56] Vipul Goyal, Yifan Song, and Chenzhi Zhu. 2020. Guaranteed Output Delivery Comes Free in Honest Majority MPC. In *CRYPTO 2020, Part II (LNCS, Vol. 12171)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 618–646. https://doi.org/10.1007/978-3-030-56880-1_22
[57] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. 2019. SoK: General Purpose Compilers for Secure Multi-Party Computation. In *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1220–1237. https://doi.org/10.1109/SP.2019.00028
[58] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. 2018. Concretely Efficient Large-Scale MPC with Active Security (or, TinyKeys for TinyOT). In *ASIACRYPT 2018, Part III (LNCS, Vol. 11274)*, Thomas Peyrin and Steven Galbraith (Eds.). Springer, Heidelberg, 86–117. https://doi.org/10.1007/978-3-030-03332-3_4
[59] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. 2018. TinyKeys: A New Approach to Efficient Multi-Party Computation. In *CRYPTO 2018, Part III (LNCS, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 3–33. https://doi.org/10.1007/978-3-319-96878-0_1
[60] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. 2017. Low Cost Constant Round MPC Combining BMR and Oblivious Transfer. In *ASIACRYPT 2017, Part I (LNCS, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Heidelberg, 598–628. https://doi.org/10.1007/978-3-319-70694-8_21
[61] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. 2020. Low Cost Constant Round MPC Combining BMR and Oblivious Transfer. *Journal of Cryptology* 33, 4 (Oct. 2020), 1732–1786. https://doi.org/10.1007/s00145-020-09355-y
[62] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2009. Secure Arithmetic Computation with No Honest Majority. In *TCC 2009 (LNCS, Vol. 5444)*, Omer Reingold (Ed.). Springer, Heidelberg, 294–314. https://doi.org/10.1007/978-3-642-00457-5_18
[63] Aayush Jain, Huijia Lin, and Amit Sahai. 2021. Indistinguishability obfuscation from well-founded assumptions. In *53rd ACM STOC*, Samir Khuller and Virginia Vassilevska Williams (Eds.). ACM Press, 60–73. https://doi.org/10.1145/3406325.3451093
[64] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2016. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 830–842. https://doi.org/10.1145/2976749.2978357
[65] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: Making SPDZ Great Again. In *EUROCRYPT 2018, Part III (LNCS, Vol. 10822)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, 158–189. https://doi.org/10.1007/978-3-319-78372-7_6
[66] Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. 2018. Accessible Privacy-Preserving Web-Based Data Analysis for Assessing and Addressing Economic Inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies* (Menlo Park and San Jose, CA, USA) *(COMPASS '18)*. Association for Computing Machinery, New York, NY, USA, Article 48, 5 pages. https://doi.org/10.1145/3209811.3212701
[67] Yehuda Lindell and Ariel Nof. 2017. A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 259–276. https://doi.org/10.1145/3133956.3133999
[68] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. 2015. Efficient Constant Round Multi-party Computation Combining BMR and SPDZ. In *CRYPTO 2015, Part II (LNCS, Vol. 9216)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.). Springer, Heidelberg, 319–338. https://doi.org/10.1007/978-3-662-48000-7_16
[69] Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. 2016. More Efficient Constant-Round Multi-party Computation from BMR and SHE. In *TCC 2016-B, Part I (LNCS, Vol. 9985)*, Martin Hirt and Adam D. Smith (Eds.). Springer, Heidelberg, 554–581. https://doi.org/10.1007/978-3-662-53641-4_21
[70] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. 2022. The Hardness of LPN over Any Integer Ring and Field for PCG Applications. Cryptology ePrint Archive, Report 2022/712. https://eprint.iacr.org/2022/712.
[71] Eleftheria Makri and Tim Wood. 2019. Full-Threshold Actively-Secure Multiparty Arithmetic Circuit Garbling. Cryptology ePrint Archive, Report 2019/1098. https://eprint.iacr.org/2019/1098.
[72] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. 2004. Fairplay - Secure Two-Party Computation System. In *USENIX Security 2004*, Matt Blaze (Ed.). USENIX Association, 287–302.
[73] Payman Mohassel and Peter Rindal. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. Cryptology ePrint Archive, Report 2018/403. https://eprint.iacr.org/2018/403.
[74] Pratyay Mukherjee and Daniel Wichs. 2016. Two Round Multiparty Computation via Multi-key FHE. In *EUROCRYPT 2016, Part II (LNCS, Vol. 9666)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, Heidelberg, 735–763. https://doi.org/10.1007/978-3-662-49896-5_26
[75] Steven Myers, Mona Sergi, and abhi shelat. 2011. Threshold Fully Homomorphic Encryption and Secure Computation. Cryptology ePrint Archive, Report 2011/454. https://eprint.iacr.org/2011/454.
[76] Jesper Buus Nielsen, Thomas Schneider, and Roberto Trifiletti. 2017. Constant Round Maliciously Secure 2PC with Function-independent Preprocessing using LEGO. In *NDSS 2017*. The Internet Society.
[77] Peter Sebastian Nordholt and Meilof Veeningen. 2018. Minimising Communication in Honest-Majority MPC by Batchwise Multiplication Verification. In *ACNS 18 (LNCS, Vol. 10892)*, Bart Preneel and Frederik Vercauteren (Eds.). Springer, Heidelberg, 321–339. https://doi.org/10.1007/978-3-319-93387-0_17
[78] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. 2020. Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption. In *CT-RSA 2020 (LNCS, Vol. 12006)*, Stanislaw Jarecki (Ed.). Springer, Heidelberg, 254–283. https://doi.org/10.1007/978-3-030-40186-3_12
[79] James S. Plank. 2007. Fast Galois Field Arithmetic Library in C/C++. http://web.eecs.utk.edu/~jplank/plank/papers/CS-07-593/
[80] Lucy Qin, Andrei Lapets, Frederick Jansen, Peter Flockhart, Kinan Dak Albab, Ira Globus-Harris, Shannon Roberts, and Mayank Varia. 2019. From Usability to Secure Computing and Back Again. Cryptology ePrint Archive, Report 2019/734. https://eprint.iacr.org/2019/734.
[81] Adi Shamir. 1979. How to Share a Secret. *Communications of the Association for Computing Machinery* 22, 11 (Nov. 1979), 612–613.
[82] Erik Taubeneck, Martin Thomson, Ben Savage, Benjamin Case, Daniel Masny, Richa Jain, Taiki Yamaguchi, Alex Koshelev, Thurston Sandbery, Victor Miller, and Shubho Sengupta. 2023. Interoperable Private Attribution (IPA).
[83] Ryan Wails, Aaron Johnson, Daniel Starin, Arkady Yerukhimovich, and S. Dov Gordon. 2019. Stormy: Statistics in Tor by Measuring Securely. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 615–632. https://doi.org/10.1145/3319535.3345650
[84] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 21–37. https://doi.org/10.1145/3133956.3134053
[85] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 39–56. https://doi.org/10.1145/3133956.3133979
[86] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th FOCS*. IEEE Computer Society Press, 162–167. https://doi.org/10.1109/SFCS.1986.25