SoK: Multiparty Computation in the Preprocessing Model

Shuang Sun   Eleftheria Makri
LIACS, Leiden University

Abstract

Multiparty computation (MPC) allows a set of mutually distrusting parties to compute a function over their inputs, while keeping those inputs private. Most recent MPC protocols that are ready for real-world applications are based on the so-called preprocessing model, where the MPC is split into two phases: a preprocessing phase, where raw material, independent of the inputs, is produced; and an online phase, which can be efficiently computed, consuming this preprocessed material, when the inputs become available. However, the sheer number of protocols following this paradigm, makes it difficult to navigate through the literature. Our work aims at systematizing existing literature, (1) to make it easier for protocol developers to choose the most suitable preprocessing protocol for their application scenario; and (2) to identify research gaps, so as to give pointers for future work. We devise two main categories for the preprocessing model, which we term *traditional* and *special* preprocessing, where the former refers to preprocessing for general purpose functions, and the latter refers to preprocessing for specific functions. We further systematize the protocols based on the underlying cryptographic primitive they use, the mathematical structure they are based on, and for special preprocessing protocols also their target function. For each of the 41 presented protocols, we give the intuition behind their main technical contribution, and we analyze their security properties and relative performance.

1 Introduction

Multiparty computation (MPC) is a mechanism that allows a set of mutually distrusting parties to compute a function over their private inputs, without revealing these inputs. More specifically, MPC allows one or more of the protocol participants to obtain the output of the function to be computed, while learning no more information about other participants' inputs, beyond what can be inferred from the protocol's output. Due to its rich functionality, allowing the computation of virtually any function on private data, MPC is a largely researched cryptographic primitive.

In 1991, the seminal work on the so-called *Beaver triples* [5] was introduced. A Beaver triple is of the form (a, b, c), with a and b uniformly random, and

$c = a \cdot b$. This is a method to compute multiplications over random values, which in turn allows for efficient multiplications over secret inputs. However, it was only in 2009 that Orlandi [74] defined an independent preprocessing phase for the costly generation of those Beaver triples, and a fast online phase to be executed when the function and inputs are known. This is, to the best of our knowledge, the foundation of the preprocessing model. The preprocessing model consists of two phases: the preprocessing phase that is independent from inputs, and the online phase that is for computing on inputs, and outputs the results. The preprocessing phase entails the evaluation of a target function on random inputs, which can be efficiently derandomized in the online phase.

Soon after the definition of the preprocessing model, thus for almost 15 years now, the preprocessing model is the most common way to design MPC protocols. This is because the preprocessing model lends itself to efficient solutions, and therefore leads to adoption of MPC in real-world applications. MPC in the preprocessing model aims at increasing the overall efficiency of MPC protocols, but the focus lies on the online phase. The goal here is to satisfy a practical need for fast evaluation of functions, once the parties' inputs are known, at the cost of a relatively slower preprocessing phase, which can take place anytime prior to the actual online execution.

In this work, we systematize MPC protocols in the preprocessing model. Firstly, we devise two main categories in this model, which we call *traditional* and *special* preprocessing, depending on the preprocessed materials they generate. We call traditional preprocessing that which generates *random values* and *Beaver triples*, enabling the sharing of private inputs, and the execution of secure multiplications in the online phase, respectively. We call special preprocessing that which provides additional preprocessed material, aimed at speeding up the online phase for specific functions (e.g., matrix multiplications). We further systematize the examined protocols, based on the underlying cryptographic primitive used in the preprocessing; based on the mathematical structure they live in; and for the special preprocessing protocols also based on their target function.

For each of the 41 protocols we analyze, we summarize their technical overview. While we strive to explain the technical contribution of each paper, our goal is to abstract it to facilitate understanding, rather than detailing concrete protocols, which can be found in the corresponding works themselves. Thus, we aim to simplify the technical description of each protocol analyzed in our SoK to highlight the core ideas and intuitions behind the technical contributions of each paper. In addition, we discuss the security assumptions of each presented protocol, and we give comparative performance metrics, in relation to prior work.

Concretely, our contributions are summarized as follows:

1. We present the first Systematization of Knowledge on MPC in the preprocessing model, an important area, with a sheer number of publications that have not been surveyed before, and we unify common terminology in this area (e.g., the preprocessing phase, aka -less accurately- offline phase);

2. We devise two main families of preprocessing protocols, based on the type of materials they generate: traditional and special preprocessing;

3. We abstract the hardcore technical details of each protocol, and focus on explaining the intuition behind each idea to facilitate understanding;

4. We identify research gaps and give pointers for future work.

1.1 Related Work

Lindell [65] presented a review article on MPC, with a clear emphasis on the security paradigm behind it. As the goal of the article is to showcase that MPC has moved from theory to real-world applications, feasibility of MPC, and use cases thereof are also an important component of this survey. The holistic (theory, practice and applications) survey on MPC by Zhao et al. [89] is motivated by the emergence of recent application scenarios, entirely matching the natural settings and requirements of MPC. This paper discusses the main underlying theoretical building blocks of MPC, the security requirements and models, and then moves to the focus points of cloud-assisted MPC, and application-oriented MPC, which are the main motivation of this article.

Vos et al. [85] focus on a specific MPC application, namely Private Set Intersection (PSI), and in particular solutions offering security against semi-honest adversaries. The proposed systematization aims at identifying research gaps, to turn protocol designers towards promising directions to solving the problem at hand efficiently. Their SoK demonstrates that older solutions that might be overlooked by new research, can still be relevant to construct efficient multiparty PSI protocols.

The survey of Zhou et al. [90] gathers the MPC-based solutions for machine learning (ML), and presents them in two categories, namely HE-based, and Secret-Sharing-based. Instead of analyzing in detail all proposed works, and their technical components, this survey focuses on identifying common challenges with respect to MPC for ML, and the adoption thereof. Then, they also provide guidelines for implementation of such systems, and propose future work, based on the currently identified limitations. Another survey focusing on a concrete application of MPC is presented by Zhang and Xin [88]. This work focuses on an active research area, namely privacy-preserving deep learning, and discusses the solutions that are based on MPC, providing a concise overview of each of the analyzed schemes.

Given the increased interest in SPDZ-like [26] MPC protocols, which is due to their good balance between security and efficiency, Orsini [75] surveys this line of work, hence covering a wide-range of concretely efficient MPC protocols with active security, in the dishonest majority setting. Another survey on concretely efficient MPC [42], goes beyond the active security with dishonest majority case that Orsini [75] analyzed, thus looking also at semi-honest adversaries, and honest majority settings; and focuses also on MPC for privacy-preserving machine learning (PPML), as well as directions for future work.

Hastings et al. [51] systematize eleven general-purpose MPC compilers. The systematization is performed on the grounds of language expressibility, cryptographic capabilities, and accessibility to developers (e.g., documentation adequacy), and based on their analysis, they provide a recommendation for each of these compiler frameworks.

Our work is the first SoK on MPC considering the preprocessing model.

2 Cryptographic Building Blocks for MPC

We assume the reader is familiar with the basic notions and primitives surrounding MPC. Here, we merely recapitulate the basic cryptographic building blocks that enable secure computation by means of MPC, and we refer the interested reader to two excellent and complementary introductions to MPC [16, 40].

Secret Sharing

Secret sharing allows a secret to be distributed among a group of parties, such that each share on its own does not reveal any information about the original secret. MPC is usually based on Linear Secret Sharing Schemes (LSSS), for which the reconstruction of the secret from the shares is a linear mapping. As such, LSSS enjoy an additive homomorphism, which allows linear operations to be performed indirectly on the secret, while actually being performed locally on the shares by each individual party, without requiring further interaction. Additive secret sharing, where the secret is simply the sum of the individual shares, and Shamir's secret sharing [84] are the two most popular choices for designing MPC protocols, and they both enjoy the property of perfect secrecy.

Oblivious Transfer

Oblivious Transfer (OT) [71, 78, 41, 54] is a protocol executed between two parties: a sender and a receiver. The most basic OT protocol is a 1-out-of-2 OT, which allows the sender to transfer one out of two possible values to the receiver. More generally, OT allows the sender to transfer some out of many pieces of information to the receiver. During an OT, the sender does not learn anything about which pieces of information were actually transferred to the receiver, and the receiver does not gain any information about the pieces they did not receive. OT is complete for MPC [62, 47]. While OT is an expensive cryptographic primitive, OT extension [53, 6] allows one to perform only a few OT's from scratch (i.e., using the standard expensive methodologies), and then be able to perform many additional OT's at the cost of a constant number of invocations of relatively inexpensive symmetric key primitives. OT extension is a fundamental building block of numerous MPC protocols.

Homomorphic Encryption

Homomorphic encryption (HE) is a form of encryption that allows computations to be performed on encrypted data without first having to decrypt it. HE comes in several flavors: Partially Homomorphic Encryption (PHE), which allows unlimited number of either additions or multiplications on the ciphertext; Somewhat Homomorphic Encryption (SHE), which allows limited number of additions and multiplications to be performed on the ciphertext; and Fully Homomorphic Encryption (FHE), which allows unlimited number of additions and multiplications to be performed on the ciphertext. PHE, notably Additively Homomorphic Encryption (AHE), aka Linearly Homomorphic Encryption (LHE), e.g., Pallier's cryptosystem [77], as well as SHE, have been extensively used in combination with MPC, or to facilitate the generation of raw material in the preprocessing phase.

3 Traditional Preprocessing

In this section, we analyze the protocols that did pioneer work, over the years, on how we generate materials in the *traditional* preprocessing phase. We reiterate that by traditional preprocessing we denote protocols that are designed to produce preprocessing material of the form of (1) *randoms*, enabling the sharing of private inputs in the online phase; (2) *triples*, allowing for efficient secure multiplication in the online phase; (3) *squares*, which are similar to the triples for computing even more efficiently a squaring in the online phase, and (4) *shared bits*, which are consumed to accelerate the online computation of non-linear functions in the online phase.

For each protocol, we explain their core technical contribution, describe their security assumptions and explain their performance, usually in terms of improvements over prior work. Table 1 summarizes our findings on traditional preprocessing protocols. Concretely, we present the reference to the corresponding work; the year of publication; the cryptographic primitive based on which the preprocessing is performed; the supported number of parties in the protocol; the security guarantees offered by the protocol (active or passive, denoted as ●, and ◐, respectively); the maximum corruption threshold; and the offline and online performance improvement over prior work (we write Comp, when the improvement refers to the computation cost, and Comm, when it refers to the communication cost). We further divide Table 1 into two sections, based on whether the protocols operate over fields, or over rings. TG denotes Triple Generation, Mul denotes Multiplication, ℓ is the length of the input, and s the statistical security parameter.

Beaver Triples. A Beaver triple is a set of secret shared values with the following special form $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, where a and b are uniformly sampled random values and c is the product of a and b. It is used for securely computing multiplications in MPC protocols. It was first proposed by Donald Beaver [5, 7], hence the name; it is also commonly called a multiplication triple.

Table 1: Overview of Protocols in the Traditional Preprocessing Model.

Work	Year	Primitive	Parties	Sec	Corruption	Offline	Online
Over Fields							
Beaver Triple [5, 7]	1991, 1998	OT	≥2	●	$<n/2$		1 round
Orlandi Protocol [74, 55]	2009, 2010	LHE	≥2	●	$n-1$	Comp: 188.4ms/TG	Comp: 15.9ms/Mul
BDOZ [9]	2011	LHE	≥2	●	$n-1$	Comp: Similar as [74]	Comp: 10^1× [74]
SPDZ [33]	2012	SHE	≥2	●	$n-1$	Comp: 13ms	Comm: $O(n^2/s)$/Mul; Comp: 0.05ms/Mul
Breaking the Limits [29]	2012	SHE	≥2	●	$n-1$	Comp: 0.5× [33]	Comp: $4.9-14.35$× [33]
TinyOT [72]	2012	OT	2	●	1	Comp: 500000 OTs/s; Overall comp: ≤ 3 sec/AES	Comm: 4 bits/gate; Comp: 20000 gates/s
Tinier [43]	2015	OT	≥ 2	●	$n-1$	Comm & Comp: 10^2 × [33]	
MASCOT [60]	2016	OT	≥2	●	$n-1$	Comm & Comp: 72× TG [33]; 200× TG [29]	
Overdrive [61]	2018	LHE	≥2	●	$n-1$	Comm: LowGear 2× TG [33]; Comp: LowGear 5.88× TG [60]	
TopGear [3]	2019	SHE	≥2	●	$n-1$	Comm & Comp: $2-5$× TG [61]	
Turbospeedz [8]	2019	LHE	≥2	●	$n-1$	Comm: Same as [76]	Comm: 2× [33]
Boyle FSS [12]	2019	FSS	2	◐	1		Comm: 10^2× [35]
Silent NISC [11]	2019	SOT	2	●	1	OT extension: up to 46.8× [53]	
Silver [24]	2021	SOT	2	●	1	Comm: Same as [11]; Comp: 19× [11]	
LowGear 2.0 [81]	2023	LHE	≥2	●	$n-1$	Comm: 1.5× [61]	
Coral [52]	2024	OT & LHE	≥2	●	$<n/2$	Comm: $2-10$× [39]; Comp: 10^1× [39]	
Over Rings							
SPDZ2k [25]	2018	OT	≥2	●	$n-1$	Comm: 0.5× [60]	
RSS19 [79]	2019	LHE	2	◐	1	Comm: 6.9× [35]; Comp: 3.6× [35]	
Overdrive2k [76]	2019	SHE	≥2	●	$n-1$	Comm: 2× [25]	
MonZa [17]	2020	LHE	2	●	1	Comm: 5.3× [25]	Same as [25]
MHz2k [21]	2021	SHE	≥2	●	$n-1$	Comm: 2.2× [17]	
TopGear2k [21]	2021	SHE	≥2	●	$n-1$	5.6× memory requirement improvement	
ACEDX21 [1]	2021	SHE RMFE	≥2	●	$<n/2$	Comm: $O(n^2/\log(n))$ ring elements/Mul	
Pika [86]	2022	FSS	≥2	●	$<n/2$	Comm & Comp: 25 × TG [27]	Comm: 74× [27]
EXY22 [39]	2022	OT RMFE	≥2	●	$n-1$	Comm: $5142.5kn(n-1)$ bits/TG	Comm: $12.4k(n-1)$ bits/Mul
Multipars [50]	2023	LHE	≥2	●	$n-1$	Comm: $8-30$× [25]; Comp: 15.2× [27]	

The main idea behind Beaver triples is to select random inputs to each gate and evaluate gates with these random values first, then correct the errors to obtain the results all at once. The correction procedure is based on a technique, which aims to prove that the product of two secrets is a third secret [4].

Consider a gate $g_k \in \{+, \times\}$ in circuit C_F. When g_k is $(\times)$, the real output value x_k can be calculated by $x_k = r_k - \Delta_k$, where r_k is a random value for the output wire of g_k and Δ_k is the correction value for it. Since the random values and correction values can be prepared in advance without any information about the real inputs, gates can later be reconstructed to obtain the result, once the input values become available. Beaver triples reduce the number of rounds of interaction by an order of magnitude.

Breaking the Limits. Damgård et al. [29] propose a SHE-based protocol that not only generates Beaver triples, but also produces "square pairs", i.e., a list of pairs of sharings $(\langle a \rangle, \langle b \rangle)$ such that $b = a^2$, and also "shared bits", i.e., a list of single shares $\langle a \rangle$ such that $a \in \{0, 1\}$. This MPC protocol can be covertly or actively secure. The notions of *square pairs* and *singles*, also appear in other earlier works, but Damgård et al. [29] are the first to formalize them.

To generate square pairs, assume each party P_i holds $\mathbf{a}_i$, a secret share of $\mathbf{a}$, and its ciphertext $c_{\mathbf{a}_i}$ is known to all parties. The parties compute $c_{\mathbf{a}^2} \leftarrow c_{\mathbf{a}} \cdot c_{\mathbf{a}}$, where $c_{\mathbf{a}} \leftarrow c_{\mathbf{a}_1} + ... + c_{\mathbf{a}_n}$. All parties execute a re-share protocol so that each party P_i obtains the share $\mathbf{b}_i$, where $\mathbf{b} = \mathbf{b}_1 + ... + \mathbf{b}_n$, and the ciphertext $c_{\mathbf{b}}$ is known to all. For the authentication, the parties compute $c_{\gamma(\mathbf{a})} \leftarrow c_{\mathbf{a}} \cdot c_\alpha$ and $c_{\gamma(\mathbf{b})} \leftarrow c_{\mathbf{b}} \cdot c_\alpha$, where α is the global MAC key. Again, parties execute the re-share protocol so that each party P_i obtains $\gamma(\mathbf{a})_i$ and $\gamma(\mathbf{b})_i$. After that, each party decomposes the various plaintext elements and obtains the secret shared square pair $(\langle a_i \rangle, \langle b_i \rangle)$.

To generate shared bits, the parties compute and decrypt $c_{\mathbf{a}^2}$ after obtaining $c_{\mathbf{a}}$, and denote the plaintext as $\mathbf{s} = \mathbf{a}^2$. If any slot position in $\mathbf{s}$ is 0, set it to 1. Then the parties take a fixed square root $\mathbf{t}$, encrypt $\mathbf{t}^{-1} \cdot c_{\mathbf{a}}$, denote as $c_{\mathbf{v}}$. The parties compute $c_{\gamma(\mathbf{v})} \leftarrow c_{\mathbf{v}} \cdot c_\alpha$, re-share it and decompose it similarly to the square pairs generation. After that, each party P_i obtains a shared bit $\langle b_i \rangle$.

The multiplication triple generation closely follows the SPDZ [33] paradigm, which we explain later in the paper. Damgård et al. [29] achieve active security by adapting the sacrificing technique to the square pairs and bit-sharings.

In the online phase, to square the sharing $\langle x \rangle$, parties take a square pair $(\langle a \rangle, \langle b \rangle)$, partially open $\langle x \rangle - \langle a \rangle$ to obtain ϵ and then each party calculates: $\langle z \rangle \leftarrow \langle b \rangle + 2 \cdot \epsilon \cdot \langle x \rangle - \epsilon^2$. The shared bits are useful in computing high level operations, such as comparisons, bit-decomposition, fixed and floating point operations.

The offline protocol has running time about twice that of SPDZ, based on Zero Knowledge Proofs of Knowledge (ZKPoKs). The SPDZ online phase has an amortized throughput of 20000 multiplications per second, over a 64-bit prime field, with 3 players. The online protocol of Damgård et al. [29] performs 98000 multiplications per second in a single thread, and 287000 with 4 threads.

3.1 Homomorphic Encryption Based Approaches

We categorize the traditional preprocessing protocols into HE-based, and OT-based, depending on the way they generate the preprocessed triples. We further systematize these works based on the underlying algebraic structure that they use, namely fields or rings.

3.1.1 Over Fields

Most of the seminal works in MPC in the preprocessing model, are based on finite fields. This came as a natural choice, due to the fact that the underlying cryptographic primitives used (e.g., Shamir, or additive secret sharing) operate over fields.

Orlandi Protocol Orlandi [74] proposes a verifiable secret sharing-based protocol for MPC over arithmetic circuits, which offers security against $n-1$ static, active corruptions. The protocol uses the circuit randomization approach of Beaver [5], but offloads the costly parts to an independent preprocessing phase, where the parties generate random shares and random triples. This does not depend on the function to be computed, nor the parties' inputs. After that, the parties use the generated triples to evaluate the circuit on their inputs.

Concretely, to generate a random share $[r]$, where r is a random value, each party P_i samples random pair $(r_i, \rho_{r,i})$ and broadcasts the commitment. Then all parties compute the commitment of (r, ρ_r), where $r = \sum_i^n r_i$ and $\rho_r = \sum_i^n \rho_{r,i}$. Denote the commitment as C_r, then each party P_i sets $[r]_i = (r_i, \rho_{r,i}, C_i)$.

Assume two random shares $[a]$ and $[b]$ are generated and given to parties. To generate a multiplication triple $([a], [b], [c])$, where $c = a{\cdot}b$, each party P_i encrypts its share a_i using its public key of a Paillier cryptosystem [77] instantiation. P_i broadcasts the encrypted share and the corresponding commitment. With the homomorphic property of the Paillier cryptosystem, the other parties can multiply their shares into the encrypted share from P_i and obtain a random share $[c]$. The product is then masked as $([c] + [r])$.

With these secret shared random values and triples, the product of two secret shared inputs $[x]$ and $[y]$ can be computed. The parties mask the inputs as $d = [x]-[a]$, $e = [y]-[b]$, and open d, e. They then compute $[z] = e[x]+d[y]-de+[c]$, hence $z = xy$. When evaluating a circuit with M multiplications with s bits of statistical security, the overhead is $O(s/\log M)$.

BDOZ. Bendlin et al. [9] propose the first practical MPC protocol for generating "standard material" in the preprocessing model, which operates over arithmetic circuits, and offers active security against a dishonest majority.

BDOZ achieves active security by computing a MAC with a random key pair $K = (\alpha, \beta)$ over $\mathbb{Z}_p^2$ to prevent parties from lying about their private shares. Concretely, the authentication code for value m is $\mathrm{MAC}_k(m) = \alpha m + \beta \bmod p$. One party holds m and $\mathrm{MAC}_k(m)$, and another party holds K. The probability of the party who holds m being able to claim a wrong value for a given MAC is $1/p$. All randoms and triples are key-consistent in this protocol, so they can be added together freely.

The offline phase achieves similar efficiency with Damgård and Orlandi's work [32]. During the online phase, BDOZ avoids the commitments for shared values, so that each party performs $O(n^2)$ multiplications to evaluate a secure multiplication. Implementation results for the two-party case indicate that each multiplication takes approximately 6 msec, which is at least an order of magnitude faster than that of Damgård and Orlandi's work on the same platform.

SPDZ. Instead of authenticating each share of secret values by using a pairwise key, as in BDOZ, Damgård et al. [33] propose a general MPC protocol, which authenticates the secret value by using a single global key. SPDZ aims to compute securely an arithmetic circuit C over any finite field $\mathbb{F}_{p^k}$ in the preprocessing model, based on SHE. The intuition and motivation behind the construction is that the authentication approach in BDOZ requires each party to have its own key, and each of the n shares needs to be authenticated with n MACs. It is easier to work with a global key, conduct some checks at the end of the protocol, and abort if malicious behavior is detected.

Assume s is a secret value, and α is the global MAC key, which is chosen randomly over $\mathbb{F}_{p^k}$. Both s and its MAC $\gamma = \alpha \cdot s$ are secret shared additively among the parties. Concretely, in the online phase, each shared value $s \in \mathbb{F}_{p^k}$ is represented as $\langle s \rangle := (\delta, (s_1, ..., s_n), (\gamma(s)_1, ..., \gamma(s)_n))$, where δ is a public value from the preprocessing phase, $s = s_1 + ... + s_n$ and $\gamma(s)_1 + ... + \gamma(s)_n = \alpha(s + \delta)$. Each party P_i holds s_i and $\gamma(s)_i$. The MAC of a value s under a global key α, is $\gamma(s)$, where $\gamma(s) \leftarrow \gamma(s)_1 + ... + \gamma(s)_n$.

To prevent a malicious adversary from inserting an error to the MACs, SPDZ introduces a sacrificing technique to check the triples before using. To check a triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, such that $c = ab + \Delta$, with Δ an error, SPDZ sacrifices another triple $(\langle f \rangle, \langle g \rangle, \langle h \rangle)$, of the same format. Let t be a random field element, then $t \cdot \Delta - \Delta'$ equals 0 with probability negligible in sec for a field of size at least 2^{sec}, if either $\Delta \neq 0$ or $\Delta' \neq 0$. This means that any error will be detected with overwhelming probability, since the parties have to reveal $[\![ta - f]\!]$ and $[\![tb - g]\!]$.

SPDZ does not check the MACs of the opened values until the output stage. Parties generate a random linear combination of both the opened value s and their shares of the corresponding MACs. They commit to the results and only then open the key α. This protocol is statistically secure against an active, adaptive corruption of $n - 1$ parties. The online phase is unconditionally secure and has total computational and communication complexity linear in n, while BDOZ has n^2. For 3 parties, a secure 64-bit multiplication can be prepared in about 13 ms in the preprocessing phase, which is 2-3 orders of magnitude faster than preliminary estimates for the most efficient instantiation of BDOZ. In the online phase, a multiplication can be done in 0.05 ms.

Overdrive. Following SPDZ [33], Keller et al. [60] proposed an MPC protocol, called MASCOT (see Section 3.2), which replaced SHE encryption with OT to do the preprocessing phase. Later, Keller et al. [61] propose a solution to preprocessing SPDZ efficiently, called Overdrive. They do the triple generation and authentication closely after MASCOT, but use lattice-based SHE instead of OT. The core of this solution is the two-party oblivious multiplication protocol by BDOZ [10]. But unlike BDOZ, Overdrive replaces the proof of correct multiplication by the SPDZ sacrifice to avoid the costliest part. The idea is that, since only the encryption of the sum of all shares is used in the protocol, Overdrive replaces the per-party proof with a global proof, where the parties prove a single joint statement together for their secret inputs and an accumulated ciphertext. This significantly reduces the computation, because every party only has to check one proof instead of $n - 1$. To further reduce the complexity of the triple generation, Overdrive generates a pair of correlated triples $((a, b, ab),$ $(a', b, a'b))$ for the sacrifice, instead of two independent ones.

Overdrive's preprocessing comes in two flavors, called LowGear and HighGear. LowGear uses the original LHE-based method of BDOZ, implemented with a level-zero LWE-based SHE scheme, namely BGV [13], which includes inherent packing. This makes LowGear very efficient for a small number of parties. For two parties, LowGear is 6× faster than MASCOT on a LAN, and 20× faster over WAN.

HighGear works better for larger values of n. It uses the original ZK proofs of SPDZ, but optimizes them by improving tightness and enabling batching. Compared to MASCOT, HighGear generates triples 6× faster, in a 2PC, 128-bit, WAN setting, and incurs the same communication.

TopGear. HighGear [61] has several limitations on the security parameters, mainly because of memory and bandwidth consumption constraints. Baum et al. [3] present a modified approach based on the ZKPoK scheme in HighGear, which they combine with the offline phase of SPDZ, resulting in the TopGear protocol. TopGear is a n-prover ZKPoK protocol, in which the n players act both as a set of provers, and individually as verifiers. TopGear provides active security against a dishonest majority.

TopGear uses the distributed decryption protocol from HighGear to obtain the MAC shares and merge them into the re-sharing scheme of Damgård et al. [29] to obtain the shares of c and a fresh encryption of c. For computing two million triples, the throughput of triples per second generated by TopGear is $2 - 5\times$ higher than by HighGear.

LowGear 2.0. LowGear 2.0 [81] introduces a new LHE-based preprocessing that removes the costly sacrificing step of Overdrive, by combining the pairwise multiplication protocol with the authentication. The core of the preprocessing is a two-party exchange protocol, which is based on the one in LowGear [61]. Assume party P_1 holds $[a]_1$, $[b]_1$, which are shares of a and b, respectively. P_2 holds $[a]_2$, $[b]_2$. P_1 encrypts $[a]_1$, $[b]_1$ with the public key and sends the ciphertexts $E([a]_1)$, $E([b]_1)$ to P_2. P_2 computes $E([a]_1)\cdot[b]_2+E([b]_1)\cdot[a]_2-E(r)$, where r is a random value, and sends it to P_1. P_1 decrypts it and obtains $d = [a]_1[b]_2 + [a]_2[b]_1 - r$. P_1 sets $[c]_1 = [a]_1[b]_1 + d$, P_2 sets $[c]_2 = [a]_2[b]_2 + r$. As such, the parties have a triple (a, b, c), where $c = [c]_1 + [c]_2 = ab$.

To guarantee the integrity of triples without sacrificing, the preprocessing protocol constructs the MAC share of $c = ab$ not from c itself, but from $\alpha c = \alpha(ab)$. Therefore, each party commits to its share of a and αb, before c and αc are constructed. To check the MACs of triples, each party samples a random value and invokes the standard authentication scheme from LowGear; then checks a linear combination of the random value and triples. LowGear 2.0

is implemented as an extension to the MP-SPDZ framework [27]. Compared to the triple generation of LowGear [61], LowGear 2.0 reduces the communication rounds from 3 to 2. The number of ciphertexts sent per ordered pair of parties is reduced by $\sim 33\%$.

3.1.2 Over Rings

As shown in Section 3.1.1, finite field arithmetic is popular in MPC. However, comparisons and bitwise operations are inefficient over finite field arithmetic, but can be greatly simplified when implemented over rings. Additionally, computing over rings makes integer arithmetic efficient, by leveraging special techniques already implemented in computers. In 2018, Cramer et al. [25] proposed SPD$\mathbb{Z}_{2^k}$, the first MPC protocol over a ring $\mathbb{Z}_{2^k}$, rather than over a finite field $\mathbb{F}_p$. This significantly simplifies implementations for comparisons and bitwise operations. However, SPD$\mathbb{Z}_{2^k}$ left an open question: can an efficient preprocessing protocol over $\mathbb{Z}_{2^k}$ be provided via HE?

RSS19. Rathee et al. [79] propose an efficient two-party protocol based on RLWE-based AHE over rings $\mathbb{Z}_{2^l}$, which is secure against a semi-honest and computationally bounded adversary. To generate n triples, party P_0 samples randomness $\langle a\rangle_0$, $\langle b\rangle_0$, and encrypts them with the public key pk: $ct_a \leftarrow Enc(pk, \langle a\rangle_0)$, $ct_b \leftarrow Enc(pk, \langle b\rangle_0)$. P_0 then sends (ct_a, ct_b) to P_1. P_1 samples randomness $\langle a\rangle_1$ and $\langle b\rangle_1$, but also another random value r. P_1 encrypts r and gets ct_r similarly to P_0. Then, P_1 computes $ct'_a \leftarrow ct_a \odot \langle b\rangle_1; ct'_b \leftarrow ct_b \odot \langle a\rangle_1; ct_d \leftarrow ct'_a \oplus ct'_b \oplus ct_r$, and sends ct_d to P_0. ct_d is decrypted by P_0 to obtain d. After all, P_0 computes $\langle c\rangle_0 \leftarrow \langle a\rangle_0 \cdot \langle b\rangle_0 + d$, and outputs $(\langle a\rangle_0, \langle b\rangle_0, \langle c\rangle_0)$. P_1 computes $\langle c\rangle_1 \leftarrow \langle a\rangle_1 \cdot \langle b\rangle_1 - r$, and outputs $(\langle a\rangle_1, \langle b\rangle_1, \langle c\rangle_1)$.

This protocol offers statistical security against a corrupted P_0 and computational security against a corrupted P_1. RSS19 outperforms the OT-based ABY [35] protocol with improvements of up to 6.9× in communication, and 3.6× in runtime for 64-bit triples generation.

Overdrive2k. Orsini et al. [76] propose Overdrive2k, a SHE-based protocol over $\mathbb{Z}_{2^k}$, which closes the performance gap between SPD$\mathbb{Z}_{2^k}$ [25] and Overdrive [61]. It provides security against $n-1$ malicious static corruptions. The online computation is performed over $\mathbb{Z}_{2^k}$, but the random authenticated data is produced over $\mathbb{Z}_{2^{(k+s)}}$, where the integer k defines the modulus 2^k, and s is a statistical security parameter. As such, the MAC Check procedure, follows closely that of SPD$\mathbb{Z}_{2^k}$ [25], but is performed over $\mathbb{Z}_{2^{(k+s)}}$.

The offline protocol generates randoms, triples, and squares similarly to SPDZ and Damgård et al. [29]. It also generates random authenticated bits for which Orsini et al. [76] propose a novel technique to implement 4-to-1 map modulo 2^t based on the 2-to-1 map modulo p, where the latter one is a standard trick for generating authenticated bits from the work of Damgård et al. [29]. The main idea behind this is temporarily working modulo 2^{t+1} and then reducing the roots modulo 2^t to obtain a 2-to-1 map. For generating a triple in the 2PC setting Overdrive2k reduces the communication to half compared to the SPD$\mathbb{Z}_{2^k}$ [25] over 64- and 128-bit data types with 64-bit statistical security. The amortized communication cost of producing triples is reduced by up to 2.6×.

MonZa. MonZa is a Joye-Libert(JL)-based [57] 2PC protocol proposed by Catalano et al. [17] for secure computation over the ring $\mathbb{Z}_{2^k}$. JL is used as the underlying AHE scheme, because it naturally supports $\mathbb{Z}_{2^k}$ as the underlying message space. In addition, JL has two important properties, all valid ciphertexts are publicly recognizable; and it provides circuit privacy for linear functions. This allows MonZa to avoid the costly ZK proofs of Overdrive2k [76]. MonZa is particularly suitable for the server-client model, where one party has less computational power than the other.

The preprocessing phase is similar to BDOZ [10] and Overdrive2k [76], but based on the JL cryptosystem. Unlike BDOZ [10] and Overdrive2k [76], MonZa's preprocessing protocol generates randoms and triples in an asymmetric way. It requires only one key pair and one party computes the intermediate ciphertext C for products of shared values, while the other party decrypts. For example, assume α the global MAC key, two parties P_1 and P_2 hold the shares α_1, α_2 respectively. To authenticate a value r known by P_1, the parties compute the shares of the product $r{\cdot}\alpha_2$. Here, P_1 encrypts r with α_1, denoted as C_1, and P_2 generates the commitment C_2 to α_2. The triple (a, b, c) can be generated by executing the above approach twice. C_1 is the encryption of a, C_2 is the commitment to b, and then this is repeated to authenticate the product c.

MonZa provides active security without the standard sacrifice step. The online phase is the same as the one of SPD$\mathbb{Z}_{2^k}$. MonZa performs better for large choices of k. More specifically, in the setting of $k = 64$, LAN latency 0.5 ms, computational security level $S = 112$, statistical security level $s = 56$, and JL modulus size = 2048-bit, the average time for triple generation on a batch of 100 runs is 52.24 ms. For $k = 128$ and $s = 40$, MonZa generates triples with $\sim 5.3\times$ better communication compared to SPD$\mathbb{Z}_{2^k}$.

MHz2k. Cheon et al. [21] propose a SHE-based MPC protocol over $\mathbb{Z}_{2^k}$, named MHz2k, actively secure, in the dishonest majority setting. The online phase follows closely that of SPD$\mathbb{Z}_{2^k}$. The triple generation and sacrifice step combine the standard methods of SPDZ [33] and Overdrive2k [76], while the authentication method is similar to the SPD$\mathbb{Z}_{2^k}$ MAC. The entire preprocessing protocol is similar to Overdirve2k, but compatible with the online protocol of SPD$\mathbb{Z}_{2^k}$ [25]. The novel insight here is that MHz2k uses a constant encoding for the ciphertext ct_α, for α the global MAC key. MHz2k proposes a Zero-knowledge Proof of Message Knowledge (ZKPoMK) to guarantee that a given ciphertext is generated with a plaintext over $\mathbb{Z}_{2^k}$. The ZKPoMK proceeds very similar to ZKPoPK, but it is more suitable for the packing method in MHz2k. Cheon et al. [21] also propose an efficient ZKPoPK over $\mathbb{Z}[X]/\phi_p(X)$, where p is a prime, named TopGear2k. This is an adaptation of TopGear to the $\mathbb{Z}_{2^k}$ case. MHz2k outperforms MonZa by 2.2×, and Overdrive2k by 3.5×, in terms of amortized communication cost of triple generation, over $\mathbb{Z}_{2^{32}}$ and $\mathbb{Z}_{2^{64}}$.

Multipars. Hasler et al. [50] present an actively secure MPC protocol over $\mathbb{Z}_{2^k}$, named Multipars. The preprocessing protocol is LHE-based and modeled following LowGear 2.0 [81]; it combines triple production and authentication to provide a triple generation protocol without the sacrificing step. Multipars successfully transferred this protocol to the $\mathbb{Z}_{2^k}$ setting by combining Overdrive2k [76], and the ZKPoMK of MHz2k [21].

Concretely, given two vectors $[\boldsymbol{a}]$ and $[\boldsymbol{b}]$, to generate an authenticated multiplication triple ($[\![\boldsymbol{a}]\!]$,$[\![\boldsymbol{b}]\!]$,$[\![\boldsymbol{c}]\!]$) that satisfies $\boldsymbol{c} = \boldsymbol{ab}$ without sacrificing, the preprocessing protocol needs to compute $[\boldsymbol{\alpha a}]$, $[\boldsymbol{\alpha b}]$, $[\boldsymbol{ab}]$ and $[\boldsymbol{\alpha ab}]$, where $\boldsymbol{\alpha}$ is the vector containing α in each entry. Assume n parties P_1, ..., P_n. For each ordered pair of parties (P_i, P_j), where P_i inputs a vector $\boldsymbol{a}$, and P_j inputs n vectors $\boldsymbol{b}_0$, $\boldsymbol{b}_1$, ..., $\boldsymbol{b}_{n-1}$. The core scheme is modeled following the LowGear pairwise multiplication protocol in Overdrive [61]. It first authenticates $[\boldsymbol{a}]$ and $[\boldsymbol{\alpha}]_i{\cdot}[\mathbf{b}]_j$. Then it multiplies $\boldsymbol{a}$ component-wise with each of the n vectors from P_j. After that, each ordered pair of parties (P_i, P_j) obtain pairwise shares $(\boldsymbol{d}^{i,j}, \boldsymbol{e}^{i,j})$ with $\boldsymbol{d}^{i,j} + \boldsymbol{e}^{i,j} = \boldsymbol{a} \odot \boldsymbol{b}^{i,j}$, which can be computed locally. Assume P_i is the prover, then P_i needs to prove that the product is a valid ciphertext. This combines authentication and component-wise multiplication and guarantees the multiplicative relation without sacrificing.

Relying on these techniques, Multipars provides protection against selective failure attacks. Regarding performance, Hasler et al. [50] present an efficient implementation of a preprocessing phase for SPD$\mathbb{Z}_{2^k}$, which computes triples up to 15.2× faster than MP-SPDZ [27]. In terms of communication per triple produced, Multipars outperforms MHz2k [21] by a factor of 2.2×, Overdrive2k [76] by around 11×, and SPD$\mathbb{Z}_{2^k}$ by $8 - 30\times$ in a two-party setup.

3.2 OT Based Approaches

3.2.1 Over Fields

Just like for traditional preprocessing based on HE primitives, OT-based approaches were also initially constructed over finite fields.

Cut&Choose. Lindell and Pinkas [66] introduce a new technique termed cut-and-choose, which combines the circuit checks for active security and the oblivious transfer.

Assume two parties P_1 and P_2, the cut-and-choose oblivious transfer is an ordinary oblivious transfer with the sender, suppose P_1, inputting many pairs (x_1^0,x_1^1),...,(x_s^0,x_s^1), with s the statistical security parameter. The receiver, suppose P_2, inputting many bits σ_1, ..., σ_s, and a set $\mathcal{J} \subset [s]$ of size exactly $s/2$. Then P_2 obtains $x_i^{\sigma_1}$ for every i along with both values (x_j^0,x_j^1) for every $j \in \mathcal{J}$, while the sender learns nothing about σ_1, ..., σ_s and $\mathcal{J}$.

Most of the exponentiation operations in this protocol can be executed in a preprocessing phase. P_1 can pre-compute its input keys independent from its actual inputs, since the garbled values of wires are all random, except the input wires. The oblivious transfers are also executed before P_2 receives its inputs. The idea is that P_2 firstly executes oblivious transfers with random inputs σ_1,...,σ_ℓ, where ℓ is the length of the inputs. After P_2 receives input bits y_1, ..., y_ℓ from P_1, it sends P_1 the correction bits $y_1 \oplus \sigma_1$,...,$y_\ell \oplus \sigma_\ell$. P_1 then exchanges the roles of P_2's two keys in input wires, for which it receives a correction bit with the value 1.

The preprocessing phase accounts for $13.5s\ell$ of the $15s\ell$ exponentiations of the protocol. Thus, the parties only need to compute $s\ell/2$ full exponentiations in the online phase. The whole protocol requires the exchange of $7s\ell+22\ell+7s+5$ group elements and has 12 communication rounds. This is a protocol to improve the general OT primitive, and not an MPC preprocessing protocol, per se. Thus, we exclude this from Table 1.

TinyOT. Nielsen et al. [72] provide an efficient implementation of a 2PC OT-based protocol for Boolean circuits. To make it secure against an active adversary, they introduce 3 approaches to put MACs on all bits, named aBit, aAND and aOT, which provide authentication for bits, local ANDs, and OTs, respectively. Both aAND, and aOT are built upon aBit, and all of them can be done in the preprocessing.

Assume two parties P_1 and P_2. To generate authenticated bits, the protocol first samples random global keys Δ_1, $\Delta_2 \in \{0,1\}^k$, held by P_2 and P_1, respectively. By using OT extension, a few seed OTs are extended into many OTs. Suppose the receiver, say P_1, wants to authenticate a bit x. It inputs x as the choice bit in an OT, while the sender, say P_2, who holds $K_x \in \{0,1\}^k$, inputs $(K_x, K_x \oplus \Delta_1)$. Then P_1 gets MAC $M_x = K_x \oplus x\Delta_1$. Let $[x]$ represent the authenticated bit x and $[x] = (x, M_x, K_x)$. P_1 can reveal $[x]$ by sending (x, M_x) to P_2. If $M_x \neq K_x \oplus x\Delta_1$, P_2 aborts. The procedure is symmetric for P_2 generating $[y]$.

Next, based on aBits, one can construct the approach aAND. One party, assume P_1, computes $c = ab$ for random a, b, then obtains an authenticated AND triple $([a],[b],[c])$ by using aBits. With these preprocessed bits and triples, the parties could compute $[z] = [x][y]$ securely. Firstly, P_1 evaluates and opens $[f] = [a] \oplus [x_1]$ and $[g] = [b] \oplus [y_1]$, where $x_1 = x \oplus x_2$, $y_1 = y \oplus y_2$. The parties then compute $[x_1 y_1] = f[y_1] \oplus g[x_1] \oplus [c] \oplus fg$. To compute $[x_2 y_2]$, the parties perform the above steps symmetrically. TinyOT lets these aBits represent the sender messages, and the receiver choice bit in a standard 1-out-2 OT. When all input and output bits are obliviously authenticated, the aOT is completed. Next, the parties compute $[s_1] = [r_2 \oplus x_2 y_1]$ and $[s_2] = [r_1 \oplus x_1 y_2]$, where $[r_1]$, $[r_2]$ are authenticated shared bits. The parties finally obtain $[z_1] = [r_1] \oplus [s_1] \oplus [x_1 y_1]$, and $[z_2] = [r_2] \oplus [s_2] \oplus [x_2 y_2]$, such that $[z] = [z_1|z_2]$.

Two aANDs and two aOTs are sufficient to evaluate any Boolean gate with only 4 bits per gate being communicated. Experiments on a device with two Intel Xeon E3430 2.40GHz cores show that the protocol generates 500000 OTs per second and can evaluate more than 20000 Boolean gates per second, for big enough circuits. Evaluating the oblivious AES encryption alone (approximately 34000 gates) takes 64 seconds, but when repeating the task 27×, it only takes less than 3 seconds per instance.

Tinier. Frederiksen et al. [43] design a set of protocols to generate binary multiplication triples, based on OT-extension. The crux of all proposed protocols in Tinier lies in authenticating (passively secure) shared values using a correlated OT extension method, which allows the adversary to introduce errors on the MACS, dependent on the MAC key. The authors show that despite the few bits leakage of the key, the protocol is actively secure. The authentication is designed by an elegant combination of the multiparty version of Tiny OT [72], introduced by Larraia et al. [64], and the standard IKNP protocol [53]. Tinier first produces unchecked triples, and afterwards performs batch verification. All proposed protocols in Tinier, producing respectively $\mathbb{F}_2$ triples, $\mathbb{F}_{2^{40}}$ triples, and MiniMac [34] ($\mathbb{F}_{2^8}$) triples perform favorably to prior work. Notably, SPDZ triples are expected to be produced faster by 2 orders of magnitude compared to the by then state-of-the-art.

MASCOT. Keller et al. [60] propose an actively secure OT-based MPC protocol for arithmetic circuits. The main goal of MASCOT is to generate triples secure against malicious adversaries in the dishonest majority setting. MASCOT achieves this by running the passively secure two-party product-sharing protocol of Gilboa [46] between every pair of parties, using k oblivious transfers to multiply two k-bit field elements. Then, they generalize the OT-based triple generation method proposed by Frederiksen et al. [43] from binary fields to finite fields, and they also closely follow their authentication method, albeit they check the MACs after opening values, by means of random linear combinations of the MACs.

MASCOT preprocessing first generates shares of correlated vector triples $(\mathbf{a}, b, \mathbf{c})$, where $b \in \mathbb{F}$ and a, $c \in \mathbb{F}^{\tau}$ for some constant τ, based on Gilboa multiplication. The parties then sample 2 public random vectors $\mathbf{r}$, $\mathbf{r}' \in \mathbb{F}^{\tau}$, and construct triples (a, b, c) and (a', b, c') with $a = \langle \mathbf{a}, \mathbf{r} \rangle$, $c = \langle \mathbf{c}, \mathbf{r} \rangle$, $a' = \langle \mathbf{a}, \mathbf{r}' \rangle$, $c' = \langle \mathbf{c}, \mathbf{r}' \rangle$. Then, the protocol adds MACs to both triples (a, b, c), (a', b, c'), and uses the standard sacrifice technique to guarantee the correctness of the triples. Privacy is ensured by producing several leaky triples and then extracting a single random triple through random combinations. The online phase is identical to SPDZ [33].

The actively secure version of MASCOT is only 6× less efficient than the passive. The triple generation of MASCOT is at least 72× faster than SPDZ [33] and 200× faster than the work of Damgård et al. [29] for a single secure multiplication, while compared to the work of Damgård et al. [28], the improvement is over 1000×.

3.2.2 Over Rings

Considering the advantages of a ring-based setting for operations, such as comparisons, OT-based approaches over rings, much like the HE-based ones have starter being introduced in the last few years.

SPDZ2k. Given that classical MAC authentication does not provide any protection over rings $\mathbb{Z}_{2^k}$, Cramer et al. [25] propose SPD$\mathbb{Z}_{2^k}$, which extends the MAC formalism of SPDZ [33] by computing MACs in some larger ring $R' = \mathbb{Z}_{2^{k+s}}$, instead of in $\mathbb{Z}_{2^k}$ directly. SPD$\mathbb{Z}_{2^k}$ is actively secure against a dishonest majority. The preprocessing is based on MASCOT, but with a different MAC procedure.

To authenticate messages, each party holds a secret shared value $[x]_i \in \mathbb{Z}_{2^t}$, such that $x = \sum_{i=1}^{n} [x]_i \bmod 2^k$ with $t = k + s$. Each party also holds a share $[\alpha]_i \in \mathbb{Z}_{2^s}$ of global MAC key α, which is sampled uniformly from $\mathbb{Z}_{2^t}$. To authenticate the shares, each party holds a value $[\gamma_x]_i$, where $\gamma_x = \sum_{i=1}^{n}[\gamma_x]_i = \alpha{\cdot}x \bmod 2^t$. An authenticated share $\langle x\rangle_i$ represents the pair of values $([x]_i,\ [\alpha x]_i)$ and is held by party P_i. Thus, to compute an arbitrary linear function, for example $y = c_0+\sum_{j=1}^{k}c_j{\cdot}x_j$, where $c_0, c_1, ..., c_k$ are public inputs, the computation goes as follows: P_1 sets $[y]_1 = c_0 + \sum_{j=1}^{k}c_j{\cdot}[x_j]_1 \bmod 2^t$; the other parties P_i, $i{\neq}1$, set $[y]_i$ similarly, but without adding c_0. Each party P_i, $i \in [n]$, calculates $[\alpha{\cdot}y]_i = [\alpha]_i{\cdot}c_0 + \sum_{j=1}^{k}c_j{\cdot}[\alpha{\cdot}x_j]_i$, then $\langle y\rangle_i$ can be represented as $([y]_i,[\alpha y]_i)$.

The communication complexity of the triple generation protocol is $O((k+s)^2)$ bits per multiplication gate. This is roughly twice the communication cost of MASCOT [60].

3.2.3 Silent OT Based Approaches

Silent preprocessing entails techniques that allow the parties in an MPC protocol to sample correlated short seeds with a small amount of interaction, and then locally (silently) expand them to long correlated randomness. “Silent” refers to the local expansion step that requires no interaction.

Silent NISC. Boyle et al. [11] propose efficient, actively secure 2-round protocols for generating useful instances of two-party correlations, such as OTs, with a small amount of communication. They propose an OT extension protocol, based on distributed point functions (DPF), which is the family of all point functions that evaluate to a non-zero value at one index in their domain. Boyle et al. [11] push the bulk of computations to an offline phase by replacing the DPF with a simpler puncturable pseudorandom function (PPRF) in the pseudorandom correlation generation (PCG) protocol. Their 2-round OT extension protocol can be applied to various traditional secure computation tasks, such as semi-honest MPC for binary circuits, and malicious MPC for binary or arithmetic circuits. It also can be used to design silent non-interactive secure computation (NISC).

The silent NISC protocol is a 2-round secure computation protocol with a silent preprocessing and a fast online phase. It works for two parties in the malicious setting. Assume functionality $f(x, y)$, where the receiver inputs x, the sender inputs y, and r is the correlated randomness. In this protocol, the receiver's piece of the correlated randomness r_R is split into two parts, r_R^{in} is used to mask the receiver's input, and r_R^{out} is used to unmask the output. The receiver is able to locally generate r_R^{in} from its public key pk_R during the preprocessing phase. In parallel, the sender also generates its own key pair $(pk_S{,}sk_S)$. The receiver then obtains r_R^{out} by using pk_S. The sender locally computes its correlated randomness r_S by invoking the OT extension protocol mentioned above.

The semi-honest NISC protocol has total communication $(2s+1){\cdot}n+O(n)$, for $O(n)$ the number of required OTs, and s the length of the string. The proposed OT extension protocol is up to 46.8× faster than the IKNP protocol [53], in a setting with 10 Mbps WAN, and approximately 5× faster in a 100 Mbps WAN.

Silver. The most costly part of Silent NISC [11] is a large matrix-vector multiplication, which requires millions of OTs. Couteau et al. [24] propose new protocols for Silent Oblivious Transfer (SOT), and vector oblivious linear evaluation (VOLE), called Silver. VOLE is a protocol that lets two parties securely compute a linear function on private input vectors without revealing these inputs. This work is not based on a well-studied assumption, it is designed upon fundamental structural properties that resist known attacks and is validated through experimental analysis.

Assume $G \in \mathbb{F}_2^{k \times n}$ is a public matrix, $\mathbf{x}$ is the vector held by the receiver in the OTs. The protocol for SOT constructs G by directly identifying its core structural properties, and allows the mapping $\mathbf{x} \rightarrow \mathbf{x} \cdot G^T$ to be computed in strict linear time. Let $\mathbf{e}$ be a uniform random vector. Silver also guarantees that distinguishing $\mathbf{e} \cdot G^T$ from random cannot be done using all known attacks on LPN and code-based cryptographic primitives. Silver is silent: after a one-time cheap interaction, two parties can store small seeds, from which they can later locally generate a large number of OTs, while remaining offline. The protocol for VOLE builds upon the protocol of Boyle et al. [11]. The receiver inputs x, the sender inputs a vector ($\mathbf{a}$, $\mathbf{b}$). The protocol allows the receiver to obtain $x \cdot \mathbf{a} + \mathbf{b}$ from the sender.

For generating 10 million random OTs on a single core of a standard laptop, Silver takes only 300ms, and 122KB of communication. Compared to Silent NISC [11], Silver incurs 19× less computation and the same communication. Compared to the IKNP [53], Silver incurs 37% less computation, and $\sim 1300\times$ less communication.

3.3 Function Dependent Preprocessing

Turbospeedz. Ben-Efraim et al. [8] propose an actively secure function-dependent preprocessing protocol for SPDZ [33], in the dishonest majority setting. By knowing the function, the preprocessing protocol is able to generate an additional secret-shared random value at each output wire of a multiplication gate, which will be used to shift the revealed values from the input wires to the output wires in the previous layer. With this, the parties compute the shares for the output wires of the following layer in the circuit.

Turbospeedz [8] introduces two protocol versions: one that uses SPDZ presprocessing as a black box, and improves the online phase; and another that modifies the Overdrive protocol [76] to improve overall communication. The latter protocol receives the circuit as input, and performs operations in the topological order of the gates of the circuit. Concretely, the parties first run the SPDZ preprocessing, and obtain the necessary number of preprocessed materials. Each multiplication gate will be associated with a Beaver triple ($\llbracket a \rrbracket$, $\llbracket b \rrbracket$, $\llbracket c \rrbracket$), a random element $\llbracket r \rrbracket$ and an additional secret-shared random value $\llbracket \lambda \rrbracket$, called the permutation element. Assume a multiplication gate with input permutation element shares $\llbracket \lambda_x \rrbracket$, $\llbracket \lambda_y \rrbracket$. The parties locally compute the shares of the offset values $\llbracket {\lambda_x}' \rrbracket \leftarrow \llbracket a \rrbracket - \llbracket \lambda_x \leftarrow$, $\llbracket {\lambda_y}' \rrbracket \leftarrow \llbracket b \rrbracket - \llbracket \lambda_y \rrbracket$, and the shares of the permutation element on the output wire $\llbracket \lambda_z \rrbracket \rightarrow \llbracket c \rrbracket - \llbracket r \rrbracket$. Parties then partially reveal the offset values $[\![\lambda_x{}']\!]$, $[\![\lambda_y{}']\!]$ for every multiplication gate, and $[\![\lambda_x{}']\!]$ for every squaring gate.

The Turbospeedz modification of Overdrive requires at most the same amount of communication as Overdrive [76] in the preprocessing. The online phase requires one value to be revealed per multiplication gate, while SPDZ [33] requires two. The MAC-Check computation cost is roughly half of SPDZ. Compared to SPDZ, Turbozpeedz improves the online communication almost by a factor of 2.

Boyle FSS. A function secret sharing (FSS) scheme is an efficient algorithm that splits a function f into two additive shares f_0, f_1, such that each of them hides f and $f_0(x) + f_1(x) = f(x)$ for every input x. Boyle et al. [12] introduce a new FSS-based 2PC protocol secure against semi-honest adversaries. The protocol is especially suitable for integer comparisons or conversions between arithmetic and boolean values, and it is generalizable to MPC. Abstractly, the protocol can be viewed as a generalization of the TinyTable protocol [31], where the novelty lies in using FSS to achieve exponential compression of the gates. This work provides a circuit-dependent approach that generates randomness aligned with the topology of the circuit, and a circuit independent approach, based on the former one.

Assume C is the circuit of f. Each gate g maps an input from a group $\mathbb{G}^{in}$ into an output from a group $\mathbb{G}^{out}$. The key idea of the FSS-based gate evaluation procedure is that, the preprocessing protocol first samples additive r offset for the input wires of g, where g is of the form $g_{r,0}(x) = g(x - r) + 0$. Then the protocol evaluates FSS shares with the FSS scheme. The outputs of the preprocessing phase are the FSS shares and $[r]$. During the online phase, the parties match up the offsets for adjacent gates, and emulate FSS shares using $[r]$, by computing $g_{[r],[r']}(x) = g(x - r) + r'$, where r' is the random mask for next gate g'. For each gate, the parties add $[r']$ to the FSS shares and then exchange the result. When the masks of the output wires are revealed, both parties can reconstruct the output.

The communication of the online phase is one element per wire. Compared to ABY [35], this work halves the online round complexity. It avoids sending a key for each bit of the input and improves the online communication by two orders of magnitude compared to ABY.

Pika. Pika [86] extends the Boyle FSS protocol [12] to work over rings. It proposes a set of FSS-based MPC protocols to securely evaluate non-linear functions such as division, exponentiation, logarithm and tanh. This work provides both semi-honest and malicious security in the honest majority setting. The structure follows prior lookup style MPC protocols. Data is secret shared between two parties P_0 and P_1, and the third party P_2 generates the FSS keys and common randomness.

The basic idea is that P_0 and P_1 locally construct a database D of the function values, such that the i-th entry $d_i = f(i)$ with i the input. This translates the problem of computing the function on the secret input into a simple database lookup. DPF outputs two vectors y_0 and y_1. To lift their shares into the ring, Pika uses a subtractive DPF, where $y_0[i] - y_1[i]$ has the value 1 or -1. Therefore, party P_2 generates shares of the DPF sign value ± 1 to avoid interactions during the translation.

Consider $[a]_{2^\ell}$ are secret shares of input a and $[b]_{2^\ell}$ secret shares of output $b = f(a)$. During the preprocessing phase, P_2 generates Beaver triples, and random values r_0 and r_1, and sends r_σ to party P_σ, where $\sigma \in \{0,1\}$. P_2 also generates FSS keys (k_0, k_1) for f_r, where f_r is the single-bit DPF at location $r \in 2^k$. P_2 generates shares (w_0, w_1) of the DPF sign bit w, and sends the tuple (k_σ, w_σ) to party P_σ. After that, P_0 and P_1 locally reconstruct $x_\sigma \equiv r_\sigma - a_\sigma \pmod{2^k}$. Each party computes y_σ with their FSS keys by invoking the DPF, sets $u_\sigma[i] = y_\sigma[i+x]$, for $i \in \{0,1,...,2^k-1\}$, and computes $v_\sigma = (-1)^\sigma \sum_{i=0}^{2^k-1} u_\sigma[i] \cdot d_i$. Then sets $v \leftarrow v + (-1)^\sigma \beta$, where β is a random value used to re-randomize v. In the end, P_0 and P_1 output the shares $[c]_{2^\ell} = [v \cdot w]_{2^\ell}$ using a Beaver triple. To achieve malicious security, SPDZ-like MACs are used.

Compared to ABY^3 [67], Pika achieves up to 128× higher throughput and 23× lower communication in the LAN setting for batch sizes of about $10^2 - 10^3$. When compared to MP-SPDZ, Pika achieves up to 25× higher throughput and 74× lower communication, in the same setting.

3.4 Quintuples

A recent, interesting line of work [1, 39, 52] studies the generation of preprocessing material, over Galois rings, based on reverse multiplication-friendly embeddings (RMFEs) [15].

ACEDX21. Guruswami and Wootters [48] showed that the Shamir's secret-sharing scheme has a regenerating property, when it is defined over an extension field, where each share can be compressed into an element using a linear form, allowing the secret to be reconstructed by a linear combination of the compressed shares. Abspoel et al. [1] propose the first concrete application of this property in MPC over Galois ring extension R/S $(\mathbb{Z}/p^k\mathbb{Z})$ of characteristic p^k. Suppose there are n parties P_1, ..., P_n, where P_i holds share x_i of secret $x \in R$, $i = 1, ..., n$. Each party P_i uses an $\mathbb{F}_2$-linear compression function $\phi_i : \mathbb{F}_{2^m} \rightarrow \mathbb{F}_2$ on its share and then sends the compressed share $\phi_i(x_i)$ to all other parties. Then each party locally reconstructs the secret $x = \sum_{i=1}^{n} \phi_i(x_i) z_i$, where $z_i \in R$ is the scalar for x_i over $\mathbb{F}_2$. Since the compressed shares do not offer error detection, this work combines Beaver triples to achieve active security. Throughout the preprocessing phase, the compressed shares $[\phi(x)]$, the scalars $[z]$ and the Beaver triple $([a],[b],[c])$ together consist a quintuple $([\phi(x)],[z],[a],[b],[c])$.

The above construction yields another protocol to compute several copies of the same circuit in parallel, secure in the honest majority setting. This protocol requires $d + O(1)$ rounds to evaluate a depth-d arithmetic circuit, with $O(n^2/log(n))$ ring elements communicated per multiplication. The communication bits for each multiplication is $\Omega(n^2)$.

EXY22. Following the work of Abspoel et al. [1], Escudero et al. [39] propose an MPC protocol over the Galois ring extension $\mathbb{Z}_{p^k}[\mathtt{X}]/h(\mathtt{X})$ of characteristic p^k that is actively secure in the honest majority setting. It also generates quintuples during the preprocessing, but in a different form: $(\langle a\rangle, \langle b\rangle, \langle\tau(a)\rangle, \langle\tau(b)\rangle, \langle\tau(a)\tau(b)\rangle)$. Among this, $a, b \in R$ are random elements, τ is a $\mathbb{Z}_{p^k}$-linear map: $R \to R$ with $\tau = \phi \circ \psi$, with ϕ defined as $\mathbb{Z}_{p^k} \to R$, ψ is $R \to \mathbb{Z}_{p^k}$.

There are two crucial points to generate this quintuple. One is generating the authenticated pairs $(\langle a\rangle, \langle\tau(a)\rangle)$. Another is obtaining the product $(\langle\tau(a)\tau(b)\rangle)$. To generate T authenticated pairs of $(\langle a\rangle, \langle\tau(a)\rangle)$, party P_i samples $a_j^i \in R$ at random for $j = 1, ..., T$, and secret shares them. P_i then performs the affine combination $r + \sum_{j=1}^{T} r_j\langle a_j^i\rangle$ to obtain $\langle\tau(a_j^i)\rangle$, $r_j \in S$. All parties then compute $\langle a_j\rangle = \sum_{i=1}^{n}\langle a_j^i\rangle; \langle\tau(a_j)\rangle = \sum_{i=1}^{n}\langle\tau(a_j^i)\rangle$. Following this, the sacrificing is performed to guarantee correctness.

The product $\langle\tau(c)\rangle = \langle\tau(a)\tau(b)\rangle$ is generated by each ordered pair of parties (P_i, P_j). Assume $\tau(a_h)^i$ and $\tau(b_h)^i$ $(h = 1, ..., N)$ are inputted to F by P_i and P_j, where F is an extended correlated oblivious product evaluation function that outputs $u_h^{(i,j)}$ to P_i and $v_h^{(j,i)}$ to P_j, such that $u_h^{(i,j)} + v_h^{(j,i)} = \tau(a_h)^i\tau(b_h)^j$. Then P_i sets its share $\tau(c_h)^i$ $\tau(c_h)^i = \tau(a_h)^i\tau(b_h)^i + \sum_{j\neq i} u_h^{(i,j)} + v_h^{(j,i)}$. Then the parties are able to compute $\langle c_h\rangle = \sum_{i=1}^{n}\langle\tau(c_h)^i\rangle$ locally. With these components, a quintuple $(\langle a\rangle, \langle b\rangle, \langle\tau(a)\rangle, \langle\tau(b)\rangle, \langle\tau(a)\tau(b)\rangle)$ is formed. Such quintuples allow us to save one online communication round per multiplication gate.

The complete preprocessing phase over $\mathbb{Z}_{p^k}$, for any prime p and integer $k \geq 1, 3$, has an average communication complexity of $5142.5kn(n-1)$, when generating one quintuple with parallel operations. This amortized online communication complexity is $12.4k(n-1)$ bits per multiplication.

Coral. The quintuple generation method from Escudero et al. [39] applies costly bucket operations to prevent leakage of $\tau(a)$ and $\tau(b)$, which leads to a cubic complexity in the bucket size. Coral. [52] is an RMFE-based MPC framework, with active security in the dishonest majority, for boolean and mixed circuits computation, overcoming this limitation.

A quintuple in Coral is represented as $([\![a]\!], [\![b]\!], [\![c]\!], [\![\tau(a)]\!], [\![\tau(b)]\!])$, where $c = \phi(\psi(a)\psi(b))$. The preprocessing generates two different kinds of triples, TinyOT and RMFE triples. For $i, j = 1, ..., (s+N)\cdot k$ with s the statistical security parameter, N the number of quintuples, the preprocessing protocol first generates TinyOT [72] triples $([\![\mathfrak{a}_i]\!]^B, [\![\mathfrak{b}_i]\!]^B, [\![\mathfrak{c}_i]\!]^B)$. Then converts them to RMFE shares $([\![\hat{a}_j]\!], [\![\hat{b}_j]\!], [\![\hat{c}_j]\!])$. After that, the protocol completes the quintuples with a standard sacrificing technique. To compute $[\![z]\!]$ such that $\psi(z) = \psi(x) \odot \psi(y)$, the online phase consumes the quintuples as $[\![z]\!] \leftarrow \tau(d)[\![\tau(b)]\!] + \tau(e)[\![\tau(a)]\!] + \tau(d)\tau(e) + [\![c]\!]$, where $[\![d]\!] \leftarrow [\![x]\!] - [\![a]\!]$, $[\![e]\!] \leftarrow [\![y]\!] - [\![b]\!]$.

Coral is implemented in MP-SPDZ [27]. Compared to EXY22, the Coral quintuple generation is over one order of magnitude faster. For input authentication, Coral reduces the communication complexity by a factor of $2 - 10\times$, when compared to that of EXY22. [39].

3.5 Degree Reduction

For Shamir's secret sharing, the degree of the random polynomial encoding of the secret goes from t to $2t$ after each multiplication. Therefore, after performing a local multiplication, MPC protocols use degree reduction schemes to restore the degree to t, to be able to continue with the computation. The degree reduction can be performed using preprocessed material, for which we present the relevant works here for completeness, but we omit these works from Table 1.

DN. Damgård and Nielsen [30] propose an actively secure multiparty computation protocol to reduce the degree of sharings from $2t$ to t, where t is the number of corrupted parties. The foundation of this work is an n-party passive secure protocol with threshold $t < n/2$. This protocol generates t-degree multiplication triples ($[a]$, $[b]$, $[c]$) during the preprocessing phase, where $c = ab$.

To reduce the degree of $[c]$ to t, each party P_i first samples a random value $s^{(i)}$ uniformly and creates both a t-sharing $[s^{(i)}]$ and a $2t$-sharing $\langle s^{(i)}\rangle$. Using a fixed matrix M, the parties compute $M\cdot([s^{(1)}],...,[s^{(n)}])$ and $M\cdot(\langle s^{(1)}\rangle,...,\langle s^{(n)}\rangle)$ to get $([r_1],...,[r_\ell])$ and $(\langle R_1\rangle,...,\langle R_\ell\rangle)$. Here, the degree of $[r_i]$ and $\langle R_i\rangle$ is t and $2t$, respectively, with $r_i = R_i$. The parties group the sharings to construct ℓ multiplication triples ($[a]$, $[b]$, ($[r]$, $\langle R\rangle$)). Next, each party computes a $2t$-sharing of D as $[a][b] + \langle R\rangle$, and they reconstruct D. After that, each party locally computes the t-sharing $[c] = D - [r]$ and outputs the multiplication triple ($[a]$, $[b]$, $[c]$), where $[c]$ also has a degree of t. Given these triples, secure multiplication can be performed during the online phase using standard methods.

The above protocol has a communication complexity of $\mathcal{O}(\mathcal{C}n)k$, where $\mathcal{C}$ is the number of gates in the circuit and k is the bitlength of the elements. It can be extended to support active security with corruption threshold $t < n/3$, and the communication complexity is $\mathcal{O}(\mathcal{C}n)k + \mathcal{O}(Dn^2)k$+poly($n\kappa$), for D the multiplicative depth of the circuit, and κ the security parameter.

Garg24. Garg et al. [45] construct a scalable MPC protocol using an unpacking approach via secret sharing based on the Chinese Remainder Theorem (CRT). The CRT-based secret sharing schemes are non-linear, but also satisfy the local homomorphism that Shamir's secret sharing does. Similar to the degree-reduction in the DN protocol [30], Garg et al. propose an integer-reduction scheme to reduce the size of the secret integer.

In the offline phase, each party P_i generates a pair of random masks ($[\![S_t^i]\!]_j$, $[\![S_{2t}^i]\!]_j$) for each other party, and sends it to them. After receiving the pairs of masks from all parties, each party P_i locally extracts $n - t$ secure masks by computing

$$\begin{pmatrix} [\![R_t^1]\!]_i \\ \vdots \\ [\![R_t^{n-t}]\!]_i \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2^\phi & 3^\phi & \cdots & n^\phi \end{pmatrix} \odot_i \begin{pmatrix} [\![S_t^1]\!]_i \\ \vdots \\ [\![S_t^n]\!]_i \end{pmatrix}$$

where $\phi = n-t-1$. ($[\![R_{2t}^1]\!]_i$,...,$[\![R_t^{n-t}]\!]_i$) can be generated with ($[\![S_{2t}^1]\!]_i$,...,$[\![S_{2t}^n]\!]_i$) in the same equation.

Similarly to the DN protocol, the parties consume a pair of masks $\{[\![R_t^i]\!], [\![R_{2t}^i]\!]\}$ per multiplication gate to reduce the size of the integer being shared. This protocol can be extended to the dishonest majority setting and achieves passive security against semi-honest corruptions for $t < 1 - \delta$ with any constant $\delta > 0$.

At a bit level, the overall communication and computation are $\mathcal{O}(|C|\cdot\log|F|)$, where $|C|$ is the circuit size and $|F|$ the field size.

4 Special Preprocessing

Special preprocessing material refers to the generation of random correlations that enable computation beyond mere multiplications. Table 2 summarizes our findings regarding preprocessing protocols targeting special functions, instead of generic computations. In Table 2, we list the name and reference of the protocols studied; the year of publication; the cryptographic primitive based on which the preprocessing is performed; the supported number of parties in the protocol; the security guarantees offered by the protocol (passive ◐ or active ●), the maximum corruption threshold; and the offline and online performance improvement over prior work, where Comm refers to communication cost, and Comp to computation cost. We further divide Table 2 into sections, based on the type of target function of the preprocessing (e.g., matrix & convolution).

4.1 Matrix Triples and Convolutions

Machine Learning (ML) usually requires a large amount of diversified data, often more than a single company can contribute. MPC offers a solution for distrustful industry competitors to train and evaluate an ML model without revealing private input data, which could, for example, contain business secrets or customer data protected by law. The large demand for privacy-preserving machine learning (PPML) technologies has also led to targeted improvements of the underlying MPC protocols. One focus of these optimizations is operations that often occur in ML tasks, like matrix multiplications or tensor convolutions, and have therefore, a large impact on the performance of a ML algorithm.

CDNN15. To securely compute linear regression models without sharing datasets between parties, Cock et al. [23] propose an MPC protocol, which aims to reduce linear regression to securely computing products of matrices. It provides information-theoretic security under the assumption of a trusted entity, and computational security using a two-party protocol, when such an entity is unavailable. The whole idea is first to have each party map their fixed precision real value inputs to elements of a finite field and create the shared matrices. Then each of them computes over their shares to obtain shares of the estimated regression coefficient vector. Lastly, parties exchange their shares of the estimated regression coefficient vector and reconstruct it.

Assume 2 shared matrices $X \in \mathbb{Z}_q^{n_1 \times n_2}$ and $Y \in \mathbb{Z}_q^{n_2 \times n_3}$, with q the size of the finite field and n_1, n_2, n_3 the dimensions of the matrices. To create the shared matrices, each party generates its share of the matrix, by mapping their respective real value inputs to elements of finite fields and putting them in the respective positions of the matrix. The remaining positions are filled with zeros; i.e., the shares X_1 and X_2 are such that $X_1 + X_2 = X$.

Table 2: Overview of Protocols in the Special Preprocessing Model.

Work	**Year**	**Primitive**	**Parties**	**Sec**	**Corruption**	**Offline**	**Online**
Matrix & Convolution							
CDNN15 [23]	2015	LHE	2	◐	1		Comp: several orders of magnitude faster [73]
SecureML [68]	2017	LHE&OT	2	◐	1	LHE Comm & Comp: 54× [73]; OT Comm & Comp: $24-1270\times$ [73]	
SecureNN [87]	2019	LHE	3	●	1	Comm and Comp: $79-553\times$ [69]	
CKRRSW20 [18]	2020	SHE	≥2	●	$n-1$	Comm: $10^2\times$ [61]	Comm: 159.2× [61]; Comp: $16-40\times$ [33]
RRHK23 [82]	2023	LHE	≥2	●	$n-1$	Comm & Comp: 4.82× [18]	Comm & Comp: 40.15× [18]
LowGear 2.0 [81]	2023	LHE	≥2	●	$n-1$	Comm: 1.64× [18]	Comp: 2.63× [18]
LookUp Tables							
TinyTable [31]	2016	OT	2	●	1		Similar as [34]
Multi-TinyTable [59]	2017	OT	≥2	●	$n-1$	Comp: 50× [31]	Comm & Comp: Same as [31]
SPOP-LUT [36]	2017	OT	2	◐	1	Comm: 1.2× [63]	Comm: 9.5× [2]
FLUTE [14]	2023	OT	2	◐	1		Comm: 100× [36]
MAESTRO [70]	2024	OT	≥2	●	$<n/2$	Comm & Comp: 1.23× [22]	Comp: 1.27× [22]
Conversion between Circuit Representations							
DaBits [83]	2019	SHE & OT	≥2	●	$n-1$	Comp: 0.5× SPDZ-based	Comp: 10× SPDZ-based
EDaBits [37]	2020	SHE & OT	≥2	●	$n-1$		Comm: $2-25\times$ [83]; Comm & Comp: 13.3× [83]
Coral [52]	2024	RMFE	≥2	●	$n/2$	Comm: 3.8× [43]; Comp: 2.5× [43]	
Tuples							
Arithmetic Tuple [80]	2022	LHE	≥2	●	$n-1$		Polynomial evaluation almost constant in the degree of the polynomial vs. logarithmic for default MP-SPDZ [58]

Let P_1 and P_2 be two parties, who aim to compute the product $Z = XY$ jointly; P_1 holds (X_1, Y_1), and P_2 holds (X_2, Y_2). Cock et al. [23] introduce a trusted entity P_0 to help with generating matrix triples to complete the task. P_0 chooses random matrices $A_1, A_2 \in \mathbb{Z}_q^{n_1 \times n_2}$, $B_1, B_2 \in \mathbb{Z}_q^{n_2 \times n_3}$, and $T \in \mathbb{Z}_q^{n_1 \times n_3}$. P_0 computes $A_1B_2 + A_2B_1$, masks it with T, and denotes the result as C. Then, it distributes (A_1, B_1, T) to P_1, and (A_2, B_2, C) to P_2.

After receiving data, P_2 masks its private shares (X_2, Y_2) by computing $(X_2 - A_2), (Y_2 - B_2)$, and sends the results to P_1. P_1 computes $A_1(Y_2 - B_2) + B_1(X_2 - A_2) + X_1Y_1$, masks it with a random matrix $T' \in \mathbb{Z}_q^{n_1 \times n_3}$, and denotes the result as W. P_1 then sends $W, (X_1 - A_1), (Y_1 - B_1)$ to P_2 and outputs $T + T'$. P_2 outputs U, where $U = (X_1 - A_1)Y_2 + (Y_1 - B_1)X_2 + X_2Y_2 + W + C$. This protocol is essentially a straightforward extension of Beaver triples for secure multiplication for matrices. Its correctness can be trivially verified by inspecting the value of $T + T' + U$.

Cock et al. [23] demonstrated the performance of their preprocessed matrix triples experimentally, and compared their work to the previous state-of-the-art [73]. CDNN15 is several orders of magnitude faster in the online phase, but the total runtime remains comparable.

SecureML. Mohassel et al. [69] proposed SecureML, a passively secure two-party system that optimizes ML operations, such as linear regression, where matrix multiplication is an essential component. In order to perform matrix multiplications, SecureML proposes two preprocessing methods to generate matrix triples: a LHE-based and an OT-based.

Assume a matrix triple $(\langle \mathbf{A} \rangle, \langle \mathbf{B} \rangle, \langle \mathbf{C} \rangle)$ is shared between two parties, P_1 and P_2. Matrices $\langle \mathbf{A} \rangle$ and $\langle \mathbf{B} \rangle$ are generated by uniformly sampling each element. Unlike Cock et al. [23], who use a trusted entity to compute the cross products $\langle \mathbf{A} \rangle_1 \langle \mathbf{B} \rangle_2$ and $\langle \mathbf{A} \rangle_2 \langle \mathbf{B} \rangle_1$, SecureML uses a LHE-based method, allowing P_1 and P_2 to compute these themselves. To generate, for instance, $\langle \mathbf{A} \rangle_1 \langle \mathbf{B} \rangle_2$, P_2 encrypts each element of $\langle \mathbf{B} \rangle_2$, and sends them to P_1, who will compute the matrix multiplication on these ciphertexts. P_1 masks the resulting ciphertexts by random values and then sends them back to P_2. P_2 decrypts these ciphertexts to obtain the resulting product. The generation of $\langle \mathbf{A} \rangle_2 \langle \mathbf{B} \rangle_1$ follows a similar procedure. After this, both parties can construct their shares of the product $\langle \mathbf{C} \rangle$.

This triple $(\langle \mathbf{A} \rangle, \langle \mathbf{B} \rangle, \langle \mathbf{C} \rangle)$ will be consumed in the online phase to mask the shares of $\langle \mathbf{X} \rangle$ and $\langle \mathbf{Y} \rangle$ for securely computing the shares of their product $\langle \mathbf{Z} \rangle_i = -i \cdot \mathbf{E} \times \mathbf{F} + \langle \mathbf{X} \rangle_i \times \mathbf{F} + \mathbf{E} \times \langle \mathbf{Y} \rangle_i + \langle \mathbf{C} \rangle_i$, where $i \in \{1, 2\}$ and $\langle \mathbf{E} \rangle, \langle \mathbf{F} \rangle$ are the masked shares. Mohassel et al. [69] experimentally show that for triple generation, the LHE-based version of SecureML has a 54× speedup compared to Nikolaenko et al. [73] in both LAN and WAN settings, and the OT-based protocol is 1270× faster in LAN, and 24× in WAN.

SecureNN. Wagh et al. [87] introduce a 3PC protocol to generate matrix triples , which completely avoids the use of garbled circuits, allowing efficient calculation of nonlinear activation functions. Their work is a combination of SecureML [69], in the sense of the straightforward extension of Beaver triples to matrix triples, and CDNN15 [23], using a trusted party for the randomness generation and distribution.

SecureNN can be tuned to provide either passive or active security, against one corrupted party. Using SecureNN for the training phase, the overall execution time is $\sim 79\times$ faster than SecureML [69] in a LAN setting, and 553× faster in a WAN setting.

CKRR20. Chen et al. [20] propose an MPC protocol for matrix multiplications and two-dimensional convolutions, which is maliciously secure, in the dishonest majority setting. Thus, the preprocessing phase generates correlated randomness in the form of matrix triples, and convolution triples.

Assume n parties wish to generate a matrix triple $([\![A]\!]_\alpha, [\![B]\!]_\alpha, [\![C]\!]_\alpha)$ with $C = AB$ and α the global MAC key. First, each party P_i samples uniformly random matrices A_i and B_i, encrypts them, and broadcasts the ciphertexts. The parties will then call the zero-knowledge proof and obtain encryptions of $A = \sum A_i$ and $B = \sum B_i$ with bounded noise. With these, the parties call the matrix multiplication algorithm of Jiang et al. [56] to compute an encryption of $C = AB$. After that, the parties can compute the encryptions of αA, αB, αC. After performing the distributed decryption, the parties get an authenticated triple $([\![A]\!]_\alpha, [\![B]\!]_\alpha, [\![C]\!]_\alpha)$.

Chen et al. [20] propose two optimizations to the preprocessing phase of SPDZ: (1) They eliminate the sacrificing step of SPDZ by switching to larger HE parameters, which supports circuits of one more depth. This leads to almost 2× improvement of the overall communication and computation. (2) They optimize the zero-knowledge proof of plaintext knowledge in the preprocessing phase of SPDZ, reducing the amortized communication overhead for proving each ciphertext from 2.5 to roughly 1.5.

Chen et al. [20] implemented their protocols and benchmarked them against Overdrive LowGear [61]. For multiplying two square matrices of size 128, CKRR20 reduced the communication cost of the preprocessing phase from 1.54 GB to 12.46 MB. In evaluating the convolution layers of the ResNet50 neural network, the online communication cost is estimated to be 159.2× lower than LowGear.

RRHK23. Although convolutions can be computed by means of matrix multiplications, this introduces additional computational overhead. To address this, Rivinius et al. [82] propose a new convolution triple generation protocol to perform convolutions directly, rather than emulating this process with matrix multiplication, as previous works do.

Assume $\boldsymbol{a}$ is a 2d image and $\boldsymbol{f}$ is a 2d filter in a convolution. A convolution triple is $([\![\boldsymbol{a}]\!]_i, [\![\boldsymbol{f}]\!]_i, [\![\boldsymbol{c}]\!]_i)$, where $\boldsymbol{a}$, $\boldsymbol{f}$ are random vectors and $\boldsymbol{c} = \text{conv2d}(\boldsymbol{a}, \boldsymbol{f})$. A convolution triple is generated follows: 1) Each party generates its share of $\boldsymbol{a}$ and $\boldsymbol{f}$ locally, encrypts one of them (i.e., $\boldsymbol{a}$) using LHE, and then sends the ciphertext to all parties; 2) Each party multiplies their own share of $\boldsymbol{f}$ with the received encrypted shares of $\boldsymbol{a}$ to obtain encrypted pairwise shares, which are re-randomized and sent back to the party that originally sent them; 3) The receiving party decrypts the pairwise shares and combines them to obtain a share of the overall product, which is the convolution of $\boldsymbol{a}$ and $\boldsymbol{f}$.

In the online phase, one can compute a convolution of a secret shared image $[\![\boldsymbol{x}]\!]_i$, and a filter $[\![\boldsymbol{y}]\!]_i$ as $[\![\mathrm{conv2d}(\boldsymbol{x}, \boldsymbol{y})]\!]_i = [\![\boldsymbol{c}]\!]_i + \mathrm{conv2d}([\![\boldsymbol{a}]\!]_i, \boldsymbol{v}) + \mathrm{conv2d}(\boldsymbol{u}, [\![\boldsymbol{f}]\!]_i) + \mathrm{conv2d}(\boldsymbol{u}, \boldsymbol{v})$, where $\boldsymbol{u} := \boldsymbol{x} - \boldsymbol{a}$ and $\boldsymbol{v} := \boldsymbol{y} - \boldsymbol{f}$ are mask values. This protocol provides active security in the dishonest majority setting. Rivinius et al. [82] implemented their work as an extension to MP-SPDZ [58]. For the evaluation of ResNet50, in a LAN setting, the preprocessing phase of RRHK23 is 4.82× faster than that of Chen et al. [20], and the online phase is up to 40.15× faster; depth-wise convolutions are up to 18.59× faster. In the WAN setting, these performance improvements are 3.01×, 41.84×, and 26.53×, respectively.

LowGear 2.0. Based on CKRR20 [19] and Overdrive LowGear [61], Hasler et al. [81] present a new solution for matrix multiplication, especially matrix squares and inner products. In order to do this, they generate matrix triples, matrix pairs during the preprocessing phase. These materials can be generated by both the exchange protocol in LowGear and their optimized version discussed in Section 3.1.1. Compared to the work of Chen et al. [20], LowGear 2.0 saves 39% in bandwidth, for matrix triple generation, in a two-party setting.

4.2 LookUp Tables

TinyTable. TinyTable [31], an optimization of MiniMac [34], is a maliciously secure 2PC, which generates random bits for masking and a scrambled version of the truth tables, in the preprocessing phase. In the online phase, parties do table lookups, using the scrambled truth tables and masked bits. While MiniMac does not work well for very tall and skinny structured circuits, TinyTable works for any circuit with the same performance as MiniMac.

Assume two parties P_1 and P_2 want to securely compute a Boolean circuit, which consists of gates $G_1, \ldots, G_N$. Let $w_1, \ldots, w_W$ be the wire labels, and $b_1, \ldots, b_W$ the actual values of the wires. For each AND gate G_i, with input wires w_u, w_v, the preprocessing phase of TinyTable operates as follows: 1) Obtains authenticated multiplication triples and random bits $[\![r_i]\!]$ for each wire, by invoking the preprocessing algorithm from TinyOT [72]; 2) Computes $[\![r_u r_v]\!]$ using the triple assigned to G_i; 3) For all $m, n \in 0, 1$, defines $t_{m,n} = (r_u + m)(r_v + n) + r_o$, where $t_{m,n}$ is the bit that needs to be additively secret shared for entry (m, n) in the table for gate G_i, and r_o is the masking bit for the output wire; 4) Computes $[\![t_{m,n}]\!] = [\![r_u r_v]\!] + m[\![r_u]\!] + n[\![r_v]\!] + [\![mn]\!] + [\![r_o]\!]$, and then P_1 sets $A_i[m, n]$ to be its share of $t_{m,n}$, where A_i is the table it holds. P_2 sets $B_i[m, n]$ similarly; 5) Opens the wire mask r_o for each output gate to both parties. Opens r_i for each input wire w_i to the party who owns this wire. The parties then return the opened masks and table A_i, B_i for each AND gate.

Multi-TinyTable. TinyTable [31] offers an efficient online evaluation, at the cost of a heavy preprocessing. To overcome this limitation, Keller et al. [59] propose a new approach, which significantly reduces the preprocessing cost, while maintaining a fast online phase. They generalize TinyTable from two-party to MPC and apply it to evaluating the S-boxes of AES, and 3DES.

To generate a masked lookup table (LUT) $([\![s]\!], [\![\mathsf{Table}(s)]\!])$, the protocol samples a random ℓ-bit mask s and bit decomposes it to $(s_0, ..., s_{\ell-1})$, where each s_i is unknown to all parties. Using these bits, the protocol expands s into a secret-shared bit vector $(s'_0, ..., s'_{2^\ell-1})$, whose s^{th} entry is 1 and is 0 elsewhere. After this, parties obtain the i^{th} entry of the masked lookup table by locally computing $[\![\mathsf{T}(s \oplus i)]\!]$. Then the preprocessing phase generates values $([\![s]\!], [\![\mathsf{Table}(s)]\!])$, where the masked table $[\![\mathsf{Table}(s)]\!]$ is of the form: $[\![\mathsf{Table}(s)]\!] = ([\![\mathsf{T}(s)]\!],\ [\![\mathsf{T}(s \oplus 1)]\!],\ ...,\ [\![\mathsf{T}(s \oplus (2^\ell - 1))]\!])$. Given such a table, evaluating $[\![\mathsf{T}(x)]\!]$ is straightforward. The parties first mask their private input x by $h = x \oplus s$ and open h. Then they locally retrieve $[\![\mathsf{Table}(s)]\!][h] = [\![\mathsf{T}(s \oplus h)]\!] = [\![\mathsf{T}(s \oplus s \oplus x)]\!] = [\![\mathsf{T}(x)]\!]$.

This protocol is actively secure in the full-threshold setting, and leads to a very fast online time of over 230000 blocks per second for AES and 45000 for 3DES. A preprocessed lookup table by means of TinyTable [31] has 4 elements, while by Multi-TinyTable [59] it has $N = 2^\ell$ elements, for ℓ the length of a random value s. For preprocessing an AES S-box over $\mathbb{F}_{2^8}$, this protocol requires 33 multiplications, more than 50 times less than TinyTable. The cost of the online phase time for evaluating a block cipher is the same as for TinyTable.

SPOP-LUT. Dessouky et al. [36] design a semi-honest secure two-party protocol that replaces the function representation of 2-input Boolean gate circuits with a LUT-based representation. This enables the evaluation of complex functions and reduces the communication complexity.

Assume two parties P_0 and P_1 have inputs x_0, $x_1 \in \{0,1\}^\delta$, respectively. and truth table $T : \{0,1\}^\delta \rightarrow \{0,1\}^\sigma$, for δ the number of inputs, and σ the number of outputs. The preprocessing is based on OT extension [63], and outputs random bits $(m_0,...,m_N)$ to the sender, say P_0, and a random choice $s \in \{0,1\}^\sigma$ to the receiver P_1. In the online phase, P_1 sends $u = s \oplus x^1$ to P_0, where $x^1 \in \{0,1\}^\sigma$ is the input of P_1. P_0 first chooses a random value z^0, computes $V = (v_0,...,v_{N-1})$, where $v_i = T[i \oplus x^0] \oplus m_{i \oplus u} \oplus z^0$, then sends V to P_1. After that, P_1 computes $z^1 = v_{x^1} \oplus m_s$.

For an AES S-Box evaluation, over WAN, SPOP-LUT reduces the communication complexity of the preprocessing by $\sim 9.5\times$ compared to Kolesnikov and Kumaresan [63], and the communication complexity of the entire protocol by 2× compared to Asharov et al. [2].

FLUTE. SPOP-LUT [36] proposes two protocol variants: one that has a good total communication, but higher online communication; and one that has better online communication, but a higher total communication. FLUTE [14] proposes a function-dependent preprocessing protocol and combines these two variants for an efficient preprocessing phase, as well as an efficient online phase. FLUTE is a two-party semi-honestly secure protocol, but can be easily extended to MPC and stronger security guarantees.

A lookup table $\mathsf{T} : \{0,1\}^\delta \rightarrow \{0,1\}^\sigma$ can be seen as a multi-input and multi-output boolean gate, which maps $\delta \geq 2$ input bits to σ output bits according to an arbitrary boolean function. The main goal of FLUTE is to convert the LUT description into a boolean expression, made up of AND and XOR gates.

During the preprocessing phase, the key step is to identify the target rows. Rather than computing LUTs by enumerating all possible inputs and outputs, FLUTE identifies only the rows that evaluate to 1, and then uses a full disjunctive normal form (DNF) representation to express the output as a function of the inputs. Assume there are α such rows. For each row, they build a term $\bigwedge_{i=1}^{\delta}\vec{\mathcal{L}}_j^i$, then set $\vec{\mathcal{L}}_j^i = x_i$ if $x_i = 1$, and set $\vec{\mathcal{L}}_j^i = \overline{x_i}$ if $x_i = 0$. The output z is represented as the OR of all such terms, $z = \bigvee_{j=1}^{\alpha}\bigwedge_{i=1}^{\delta}\vec{\mathcal{L}}_j^i$. Since two different terms of the form $\bigwedge_{i=1}^{\delta}\vec{\mathcal{L}}_j^i$ and $\bigwedge_{i=1}^{\delta}\vec{\mathcal{L}}_{j'}^i$ can never both evaluate to 1, one can directly substitute OR operations with XOR operations.

Compared to SPOP-LUT [36], FLUTE improves the online communication by more than 100×, while keeping the increase in total communication overhead to less than 4% on average.

MAESTRO. MAESTRO [70] is a set of MPC protocols focusing on the computation of the S-box of AES, in the honest majority setting, with both semi-honest and maliciously secure variants. The main protocol in this work is a secure protocol for computing the multiplicative inverse with a lookup table of size 16. The core idea is to reduce the computation of multiplicative inversion over $GF(2^8)$ into the one over $GF(2^4)$. This can be achieved by using the isomorphism between $GF(2^8)$ and $GF((2^4)^2)$. MAESTRO provides a lookup table protocol that takes a secret sharing $[\![v]\!]$, $v \in GF(2^4)$ and a table T as inputs, and outputs the corresponding shared value $[\![T_v]\!]$. When T is an inversion table, the outputs will be $T_x = v^{-1}$. The preprocessing generates shared randomness $[\![r]\!]$ and its corresponding shared random one-hot vector $[\![\mathbf{e}^{(r)}]\!]$, where the r^{th} element is 1, and all others 0. In the online phase, the input is masked with $[\![r]\!]$, and the masked value c, is opened. After that, the table lookup can be performed as a linear function on $\mathbf{e}^{(r)}$. The parties then compute $[\![t]\!] := \bigoplus_{j=0}^{N-1}[\![\mathbf{e}_j^{(r)}]\!]\cdot T_{c\oplus j}$.

In the 3PC, semi-honest setting, MAESTRO is faster than the work of Chida et al. [22], by 27% for the online phase, and by 23% for total throughput. For active security, this work improves the total throughput by $46\% - 270\%$ in LAN and by up to 453% in WAN, compared with the work of Chida et al. [22], and Furukawa et al. [44].

4.3 Conversion between Circuit Representations

DaBits. In 2019, Rotaru and Wood proposed a novel type of preprocessing material, called daBits [83]. Using a preprocessed daBit, which stands for doubly-shared authenticated bit, one can easily implement a conversion of their computation from Boolean to arithmetic circuits and vice versa, in the online phase. The preprocessed daBit is essentially a random bit secret shared both in $\mathbb{F}_p$ and in $\mathbb{F}_2$. Using this preprocessed bit in the online phase to mask an input, together with the simple property of XOR in any field: $\mathrm{XOR}(x, y) \mapsto x+y-2\cdot x\cdot y$ one can mask a value in the origin space, open it, and derandomize it in the target space. daBits [83] offer active security guarantees in the full-threshold setting, while working for any number of parties, and any generic MPC based on linear secret sharing. The authors report being able to evaluate a linear Support Vector Machine with 400 fewer AND gates than the generic GC approach, at the cost of doubling the required preprocessing compared to merely using SPDZ.

EDaBits. Escudero et al. [38] propose a new approach, which generates extended daBits (edaBits).Unlike a daBit, which is a bit shared in two different domains, an edaBit consists of a set of random bits $(r_{m-1}, ..., r_0)$, each bit is secret shared as $[r_i]_2$ in the binary domain and an integer $r = \sum_{i=0}^{m-1} r_i 2^i$, which is secret-shared as $[r_i]_M$ in the arithmetic domain. An m-length edaBit can be generated from m daBits, but the direct edaBits generation is much more efficient. Each party first samples a value $r^i \in \mathbb{Z}_M$ privately and secret shares it. Next, it bit-decomposes r^i, and secret shares the bits in the binary domain.

EdaBits provide active security in the full-threshold setting. For secure comparisons, edaBits improve $2 - 25\times$ the communication cost compared to daBits, and the number of comparisons per second is increased by up to $13.3\times$.

Coral. Coral [52] presents a new RMFE-based mixed-circuit evaluation method by vectorizing the daBit [83] and edaBit [38] generation methods into RMFE shares. The main idea is to construct global packed edaBits from private ones and correct the arithmetic part using the RMFE-based boolean-to-arithmetic functionality, which can be implemented with packed daBits.

Let $\llbracket \mathbf{r} \rrbracket^A$ denote a vector of authenticated arithmetic sharings in $\mathbb{Z}_q$. The packed daBits can be represented as $(\llbracket \mathbf{r} \rrbracket^A, \llbracket r \rrbracket)$, where $\mathbf{r} \in \{0,1\}^k$ and $r = \phi(\mathbf{r})$, $\phi : \mathbb{Z}_q^k \to \mathbb{Z}_q$. They are generated with a vectorized version of edaBits. Then the packed daBits will be consumed to convert a RMFE sharing into a vector of arithmetic sharings. The consumption procedure in the online phase, is similar to the standard flow from Damgård et al. [27], which is designed for traditional boolean sharings. Coral then constructs the RMFE-based edaBit $(\llbracket \mathbf{r} \rrbracket^A, \llbracket r_0 \rrbracket, ..., \llbracket r_{\ell-1} \rrbracket)$, where ℓ is the length of the edaBit, $\mathbf{r} \in \{0,1\}_q^k$ and $\mathbf{r}[i] = \sum_{j=0}^{\ell-1} \psi(r_j)[i] \cdot 2^j$. This packed edaBit is equivalent to k plain edaBits.

The edaBit generation by Coral, in a WAN setting, gives $4\times$ higher throughput than Tinier [43], when both are paired with LowGear for the arithmetic protocol. Loose edaBits generation by Coral offers 73.6% communication reduction compared to Tinier, and $2.5\times$ speedup, in the single-threaded WAN setting.

4.4 Tuples

Arithmetic Tuples. Binomial tuples can reduce the round complexity and bandwidth compared to using Beaver triples, but have exponential size in the number of inputs. Reisert et al. [80] generalize both Beaver triples and binomial tuples, proposing a new form of correlated randomness, called arithmetic tuples. Using arithmetic tuples, they propose an actively secure protocol that allows to evaluate any multivariate polynomial in one round of online communication plus one opening round, with a moderate tuple size.

During the preprocessing phase, a sufficient number of Beaver triples is generated. Using these Beaver triples, the preprocessing phase then runs the SPDZ online protocol to compute the tuples. For each party P_i, who holds a secret shared and authenticated input $\llbracket x \rrbracket_i$, the preprocessing protocol samples a set of random masks (a_1,...,a_l), computes the tuple (a_1,...,a_k), where $l < k$. The protocol then authenticates the tuple and sends $[\![(a_1, ..., a_k)]\!]$ to P_i. For example, to compute the product of four shared inputs $[\![x_0]\!] \cdot [\![x_1]\!] \cdot [\![x_2]\!] \cdot [\![x_3]\!]$, the preprocessing protocol generates 13-tuple for each party with the form ($[\![a_0]\!]$, $[\![a_1]\!]$, $[\![a_2]\!]$, $[\![a_3]\!]$, $[\![a_0a_1]\!]$, $[\![a_{01}]\!]$, $[\![a_{01}a_2]\!]$, $[\![a_{01}a_3]\!]$, $[\![a_2a_3]\!]$, $[\![a_{23}]\!]$, $[\![a_{23}a_0]\!]$, $[\![a_{23}a_1]\!]$, $[\![a_{01}a_2a_3 + a_{23}a_0a_1 - a_{01}a_{23}]\!]$).

During the online phase, each party P_i computes and opens $[\![x_j]\!]_i - [\![a_j]\!]_i$ for all $0 \leq j < 4$. Then each party P_i computes locally and opens $[\![y_{01}]\!]$, $[\![y_{23}]\!]$, $[\![y_{0123}]\!]_i$, where

$$
\begin{aligned}
&[\![y_{01}]\!] = (x_0 - a_0)[\![x_1]\!]_i + [\![a_0]\!]_i(x_1 - a_1) + [\![a_0a_1]\!]_i - [\![a_{01}]\!]_i;\\
&[\![y_{23}]\!] = (x_2 - a_2)[\![x_3]\!]_i + [\![a_2]\!]_i(x_3 - a_3) + [\![a_2a_3]\!]_i\\
&- [\![a_{23}]\!]_i;\\
&[\![y_{0123}]\!]_i = (x_0 - a_0)(x_1 - a_1)[\![a_{23}]\!]_i + (x_0 - a_0)[\![a_1a_{23}]\!]_i\\
&+ (x_1 - a_1)[\![a_0a_{23}]\!]_i + (x_2 - a_2)[\![a_3a_{01}]\!]_i\\
&+ (x_3 - a_3)[\![a_2a_{01}]\!]_i + (x_2 - a_2)(x_3 - a_3)\\
&[\![a_{01}]\!]_i + [\![a_0a_1a_{23} + a_{01}a_2a_3 - a_{01}a_{23}]\!]_i.
\end{aligned}
$$

The protocol first computes x_0x_1 and x_2x_3, masks them with fresh randomness a_{01}, a_{23} respectively. Then the protocol computes the product of all four variables. Note that, the masking operations lead to a mixed term $a_{23}x_0x_1 + a_{01}x_2x_3 - a_{01}a_{23}$ in the second level multiplication, it will be removed with y_{0123}. The security of this method follows analogously to Beaver multiplication.

Since the focus of this work is on applications where the offline phase is not time-critical, Reisert et al. [80] implemented the online phase in the MP-SPDZ framework [58]. In terms of polynomial evaluation, this work requires lower bandwidth and an almost constant runtime compared to the default MP-SPDZ implementation.

5 Discussion

When studying such a large body of research, it is surprising to notice that several core ideas are long present in the field of MPC, but they only get formalized or exploited for practical efficiency gains many years later. For instance, the idea of commodity-based MPC dates back to the seminal works of Beaver [5, 7], but it took the research community almost 1, 5 decades to formalize the preprocessing model, which is a prime example of commodity-based MPC. This shows that important areas of research can greatly benefit from systematization, in order to reveal such (hidden) ideas, organize existing work in meaningful categories, and identify gaps to propose directions for future research and applications.

Protocols that fall within what we termed traditional preprocessing model are the foundation of this research area, and they will remain important, as they enable the secure computation of general purpose functions. As such, in their generality, and universal applicability, they remain relevant and useful for practical applications. Yet, our systematization confirmed that MPC in the special preprocessing model, is strictly more efficient for special functions, than designing secure computation protocols in a straightforward manner from generic traditional preprocessing.

Systematizing the seminal works in the traditional preprocessing model shows that most of the protocols offer active security guarantees, in the full-threshold setting. This is a realistic assumption in many application scenarios, especially considering the balance between security and efficiency that this line of work offers. However, when the application scenario at hand mandates more stringent security guarantees to be in place, we identify that there is a research gap here. It is therefore an interesting area of investigation for future work, to consider constructions in the preprocessing model, which offer higher security guarantees, such as identifiable abort, fairness, or guaranteed output delivery, in an efficient manner.

Recently, Hamilis et al. [49] aim at producing "preprocessing [material] for life", as their title implies. This work not only envisions a one-time setup enabling consequently efficient online computations, but it even extends the security assumptions, to achieve identifiable abort. Addressing some of the limitations of this work, such as the asymmetry between the parties' computational and communication power in the preprocessing phase, serves as an excellent basis for future work. Considering preprocessing for life, also begs the question as to how we can design preprocessing protocols for dynamic sets of parties.

It is also evident from our systematization that the work over rings is much younger than the work over fields, with even more recent advances (about yet a decade later) over ring extensions leveraging RMFEs. Given the natural fit of such constructions for computer systems, as well as for operations that have been traditionally the bottleneck of secure MPC, such as comparisons, further study of ring-based preprocessing protocols is an interesting future research prospect. The fact that work over rings is so much younger than preprocessing over fields, also explains why there is so little work in the special preprocessing model that leverages such mathematical structures. Thus, we identify special preprocessing over rings as another interesting area for future research.

The attention towards the special preprocessing model is increasing in the last few years. Our systematization shows that both the very first attempts to do special preprocessing, as well as some of the most recent results evolve around preprocessing material that enables machine learning applications (e.g., matrices and convolutions). Given the current trend of privacy-preserving machine learning, efficiently producing special preprocessing material for such tasks is expected to remain relevant.

References

[1] Mark Abspoel, Ronald Cramer, Daniel Escudero, Ivan Damgård, and Chaoping Xing. Improved single-round secure multiplication using regenerating codes. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in*

Cryptology – ASIACRYPT 2021, Part II, volume 13091 of *Lecture Notes in Computer Science*, pages 222–244, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 535–548, Berlin, Germany, November 4–8, 2013. ACM Press.

[3] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*, volume 11959 of *Lecture Notes in Computer Science*, pages 274–302, Waterloo, ON, Canada, August 12–16, 2019. Springer, Cham, Switzerland.

[4] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, January 1991.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Berlin, Heidelberg, Germany.

[6] Donald Beaver. Correlated Pseudorandomness and the Complexity of Private Computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488, 1996.

[7] Donald Beaver. One-Time Tables for Two-Party Computation. In *Computing and Combinatorics: 4th Annual International Conference COCOON'98 Taipei, Taiwan, RoC, August 12–14, 1998 Proceedings 4*, pages 361–370. Springer, 1998.

[8] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*, volume 11464 of *Lecture Notes in Computer Science*, pages 530–549, Bogota, Colombia, June 5–7, 2019. Springer, Cham, Switzerland.

[9] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semihomomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Heidelberg, Germany.

[10] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic Encryption and Multiparty Computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–188. Springer, 2011.

[11] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 291–308, London, UK, November 11–15, 2019. ACM Press.

[12] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 341–371, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[14] Andreas Brüggemann, Robin Hundt, Thomas Schneider, Ajith Suresh, and Hossein Yalame. FLUTE: Fast and secure lookup table evaluations. In *2023 IEEE Symposium on Security and Privacy*, pages 515–533, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.

[15] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 395–426, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[16] Dario Catalano, Ronald Cramer, Giovanni Di Crescenzo, Ivan Darmgård, David Pointcheval, and Tsuyoshi Takagi. Multiparty Computation, an Introduction. *Contemporary cryptology*, pages 41–87, 2005.

[17] Dario Catalano, Mario Di Raimondo, Dario Fiore, and Irene Giacomelli. Mon$\mathbb{Z}_{2^k}$a: Fast maliciously secure two party computation on $\mathbb{Z}_{2^k}$. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 357–386, Edinburgh, UK, May 4–7, 2020. Springer, Cham, Switzerland.

[18] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. Cryptology ePrint Archive, Report 2020/451, 2020.

[19] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III 26*, pages 31–59. Springer, 2020.

[20] Hao Chen, Miran Kim, Ilya P. Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 31–59, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

[21] Jung Hee Cheon, Dongwoo Kim, and Keewoo Lee. MHz2k: MPC from HE over $\mathbb{Z}_{2^k}$ with new packing, simpler reshare, and better ZKP. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 426–456, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[22] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, and Benny Pinkas. High-throughput secure AES computation. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 13–24, 2018.

[23] Martine de Cock, Rafael Dowsley, Anderson CA Nascimento, and Stacey C Newman. Fast, Privacy Preserving Linear Regression over Distributed Datasets Based on Pre-distributed Data. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2015.

[24] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 502–534, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[25] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 769–798, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[26] I. Damgard, V. Pastro, N.P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011.

[27] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy*, pages 1102–1120, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

[28] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 241–263, Amalfi, Italy, September 5–7, 2012. Springer, Berlin, Heidelberg, Germany.

[29] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18, Egham, UK, September 9–13, 2013. Springer, Berlin, Heidelberg, Germany.

[30] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Berlin, Heidelberg, Germany.

[31] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 167–187, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[32] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 558–576, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Berlin, Heidelberg, Germany.

[33] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[34] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of Boolean circuits using preprocessing. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes*

in Computer Science, pages 621–641, Tokyo, Japan, March 3–6, 2013. Springer, Berlin, Heidelberg, Germany.

[35] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, February 8–11, 2015. The Internet Society.

[36] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *ISOC Network and Distributed System Security Symposium – NDSS 2017*, San Diego, CA, USA, February 26 – March 1, 2017. The Internet Society.

[37] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. Cryptology ePrint Archive, Report 2020/338, 2020.

[38] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 823–852, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[39] Daniel Escudero, Chaoping Xing, and Chen Yuan. More efficient dishonest majority secure computation over $\mathbb{Z}_{2^k}$ via galois rings. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 383–412, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[40] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A Pragmatic Introduction to Secure Multi-Party Computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

[41] Shimon Even, Oded Goldreich, and Abraham Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[42] Dengguo Feng and Kang Yang. Concretely Efficient Secure Multi-Party Computation Protocols: Survey and More. *Security and Safety*, 1:2021001, 2022.

[43] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 711–735, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Berlin, Heidelberg, Germany.

[44] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 225–255, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland.

[45] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, and Mingyuan Wang. Scalable multiparty computation from non-linear secret sharing. In *Annual International Cryptology Conference*, pages 384–417. Springer, 2024.

[46] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 116–129, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Heidelberg, Germany.

[47] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play any Mental Game, or a Completeness Theorem for Protocols with Honest Majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. 2019.

[48] Venkatesan Guruswami and Mary Wootters. Repairing reed-solomon codes. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing*, pages 216–226, Cambridge, MA, USA, June 18–21, 2016. ACM Press.

[49] Matan Hamilis, Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Preprocessing for Life: Dishonest-Majority MPC with a Trusted or Untrusted Dealer. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 41–41. IEEE Computer Society, 2024.

[50] Sebastian Hasler, Pascal Reisert, Marc Rivinius, and Ralf Küsters. Multipars: Reduced-Communication MPC over Z2k. *Cryptology ePrint Archive*, 2023.

[51] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. SoK: General Purpose Compilers for Secure Multi-Party Computation. In *2019 IEEE symposium on security and privacy (SP)*, pages 1220–1237. IEEE, 2019.

[52] Zhicong Huang, Wen-jie Lu, Yuchen Wang, Cheng Hong, Tao Wei, and WenGuang Chen. Coral: Maliciously Secure Computation Framework for Packed and Mixed Circuits. *Cryptology ePrint Archive*, 2024.

[53] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Heidelberg, Germany.

[54] Yuval Ishai and Eyal Kushilevitz. Private Simultaneous Messages Protocols with Applications. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*, pages 174–183. IEEE, 1997.

[55] Thomas P. Jakobsen, Marc X. Makkes, and Janus Dam Nielsen. Efficient implementation of the Orlandi protocol. In Jianying Zhou and Moti Yung, editors, *ACNS 10: 8th International Conference on Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 255–272, Beijing, China, June 22–25, 2010. Springer, Berlin, Heidelberg, Germany.

[56] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1209–1222, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[57] Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 76–92, Athens, Greece, May 26–30, 2013. Springer, Berlin, Heidelberg, Germany.

[58] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1575–1590, Virtual Event, USA, November 9–13, 2020. ACM Press.

[59] Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Eduardo Soria-Vazquez, and Srinivas Vivek. Faster secure multi-party computation of AES and DES using lookup tables. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 229–249, Kanazawa, Japan, July 10–12, 2017. Springer, Cham, Switzerland.

[60] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.

[61] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822

of *Lecture Notes in Computer Science*, pages 158–189, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[62] Joe Kilian. Founding Cryptography on Oblivious Transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, 1988.

[63] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 54–70, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Heidelberg, Germany.

[64] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 495–512, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Heidelberg, Germany.

[65] Yehuda Lindell. Secure Multiparty Computation. *Communications of the ACM*, 64(1):86–96, 2020.

[66] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346, Providence, RI, USA, March 28–30, 2011. Springer, Berlin, Heidelberg, Germany.

[67] Payman Mohassel and Peter Rindal. ABY^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 35–52, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[68] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. Cryptology ePrint Archive, Report 2017/396, 2017.

[69] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[70] Hiraku Morita, Erik Pohle, Kunihiko Sadakane, Peter Scholl, Kazunari Tozawa, and Daniel Tschudi. MAESTRO: Multi-party AES using Lookup Tables. *Cryptology ePrint Archive*, 2024.

[71] Moni Naor and Benny Pinkas. Oblivious Transfer and Polynomial Evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254, 1999.

[72] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Heidelberg, Germany.

[73] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[74] Claudio Orlandi. LEGO and other cryptographic constructions. Technical report, Citeseer, 2009.

[75] Emmanuela Orsini. Efficient, Actively Secure MPC with a Dishonest Majority: A Survey. In *Arithmetic of Finite Fields: 8th International Workshop, WAIFI 2020, Rennes, France, July 6–8, 2020, Revised Selected and Invited Papers 8*, pages 42–71. Springer, 2021.

[76] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, volume 12006 of *Lecture Notes in Computer Science*, pages 254–283, San Francisco, CA, USA, February 24–28, 2020. Springer, Cham, Switzerland.

[77] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Berlin, Heidelberg, Germany.

[78] Michael O Rabin. How to Exchange Secrets with Oblivious Transfer. *Cryptology ePrint Archive*, 2005.

[79] Deevashwer Rathee, Thomas Schneider, and K. K. Shukla. Improved multiplication triple generation over rings via RLWE-based AHE. In Yi Mu, Robert H. Deng, and Xinyi Huang, editors, *CANS 19: 18th International Conference on Cryptology and Network Security*, volume 11829 of *Lecture Notes in Computer Science*, pages 347–359, Fuzhou, China, October 25–27, 2019. Springer, Cham, Switzerland.

[80] Pascal Reisert, Marc Rivinius, Toomas Krips, and Ralf Kuesters. Arithmetic tuples for MPC. Cryptology ePrint Archive, Report 2022/667, 2022.

[81] Pascal Reisert, Marc Rivinius, Toomas Krips, and Ralf Küsters. Overdrive LowGear 2.0: Reduced-bandwidth MPC without sacrifice. In Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik, editors, *ASIACCS 23: 18th ACM Symposium on Information, Computer and Communications Security*, pages 372–386, Melbourne, VIC, Australia, July 10–14, 2023. ACM Press.

[82] Marc Rivinius, Pascal Reisert, Sebastian Hasler, and Ralf Kuesters. Convolutions in overdrive: Maliciously secure convolutions for MPC. Cryptology ePrint Archive, Report 2023/359, 2023.

[83] Dragos Rotaru and Tim Wood. Marbled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security. In *International Conference on Cryptology in India*, pages 227–249. Springer, 2019.

[84] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[85] Jelle Vos, Mauro Conti, and Zekeriya Erkin. SoK: Collusion-Resistant Multi-Party Private Set Intersections in the Semi-Honest Model. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 465–483. IEEE, 2024.

[86] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies*, 2022(4):351–377, October 2022.

[87] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, July 2019.

[88] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. Privacy-Preserving Deep Learning based on Multiparty Secure Computation: A Survey. *IEEE Internet of Things Journal*, 8(13):10412–10429, 2021.

[89] Chuan Zhao, Shengnan Zhao, Minghao Zhao, Zhenxiang Chen, Chong-Zhi Gao, Hongwei Li, and Yu-an Tan. Secure Multi-Party Computation: Theory, Practice and Applications. *Information Sciences*, 476:357–372, 2019.

[90] Ian Zhou, Farzad Tofigh, Massimo Piccardi, Mehran Abolhasan, Daniel Franklin, and Justin Lipman. Secure Multi-Party Computation for Machine Learning: A Survey. *IEEE Access*, 2024.