Efficient, Actively Secure MPC with a Dishonest Majority: a Survey

Emmanuela Orsini
imec-COSIC, KU Leuven
emmanuela.orsini@kuleuven.be

Abstract

The last ten years have seen a tremendous growth in the interest and practicality of secure multiparty computation (MPC) and its possible applications. Secure MPC is indeed a very hot research topic and recent advances in the field have already been translated into commercial products world-wide. A major pillar in this advance has been in the case of active security with a dishonest majority, mainly due to the SPDZ-line of work protocols. This survey gives an overview of these protocols, with a focus of the original SPDZ paper (Damgård et al. CRYPTO 2012) and its subsequent optimizations. It also covers some alternative approaches based on oblivious transfer, oblivious linear-function evaluation, and constant-round protocols.

1 Introduction

Secure Multiparty Computation (MPC) allows a set of parties to compute a joint function on their inputs while maintaining *privacy*, meaning that the output of the computation should not reveal anything but the output itself.

The concept of secure computation was introduced by Andrew Yao [Yao86] who presented a two-party protocol for Boolean circuits based on the idea of *garbled circuits*. Yao's protocol is a constant-round protocol, where one party, the garbler, generates an encrypted version of the circuit that is securely evaluated by the other party, the evaluator. After forty years this protocol still remains the basis for many efficient MPC implementations.

After Yao's garbled-circuit based protocol was proposed, several multiparty protocols appeared both for Boolean and arithmetic circuits, including those given by Goldreich, Micali and Wigderson (GMW) [GMW87] , Ben Or, Goldwasser and Wigderson (BGW) [BGW88], Chaum, Crepeau and Damgård (CCD) [CCD88]. All of these protocols have a number of rounds linear in the depth of the circuit to be evaluated and consist in evaluating the circuit gate-by-gate using a secret-sharing of the data. In 1990, Beaver, Micali and Rogaway presented the BMR protocol [BMR90] generalizing Yao's approach to the multiparty setting.

The BMR protocol runs in a constant number of rounds, while achieving security against dishonest majority. Almost all known secure MPC protocols rely on techniques described in these fundamental works.

Secure multiparty computation should guarantee a number of security properties other than privacy, even in the presence of some adversarial entity that controls a subset of the parties, usually referred to as *corrupt* parties. The most significant of these properties are: a) *correctness*, meaning that each party should receive a correct output; b) *independence of the inputs*, i.e. corrupt parties' inputs should be independent of honest parties' inputs; c) *guaranteed output delivery*, namely honest parties should always be able to receive their outputs; d) *fairness*, i.e. corrupt parties should receive their outputs if and only if honest parties do. Note that fairness is a weaker requirement than guaranteed output delivery; indeed guaranteed output delivery implies fairness, but the opposite is not always true [CL14].

Secure multiparty computation comes in different flavors according to diverse corruption strategies, security requirements, model of computation, communication channels, etc. However, not all the possible combinations of these properties and settings are possible. One major distinction is between protocols that rely on the existence of a *honest majority*, i.e. less than a half of the total number of parties is corrupt, and protocols that can be proven secure even with no honest majority. In the first case it is possible to describe unconditionally secure protocols, whereas to deal with a dishonest majority it is necessary to restrict to computational security that holds under some cryptographic assumption. Note that assuming a honest majority is sometimes a strong requirement, and it is pointless in the important case of two-party computation.

Another crucial division is determined by the type of corruptions that the protocol can support. There are three main adversary models that are usually considered: 1) *semi-honest/passive adversary*, that follows the protocol specifications but tries to gain more information than what is allowed; 2) *malicious/active adversary*, that can arbitrarily deviate from the protocol in order to break the inputs' privacy and/or the outputs' correctness; 3) *covert adversary* that may behave maliciously, but with a fixed probability to be spotted. While actively secure protocols are always able to detect malicious corruptions (but not necessarily the identity of corrupt parties), in covert secure protocols a cheating party might not be detected with a certain non-negligible probability. Semi-honest protocols offer a rather weak security guarantee, but they are much more efficient than maliciously secure protocols. Interestingly, covert security can be thought as a compromise between the other two more standard models, as it can offer more efficiency than active security and stronger guarantees than semi-honest one.

Besides this efficiency issue and the need of cryptographic assumptions, Cleve in [Cle86] showed that in the very desirable setting of active security and dishonest majority it is impossible to obtain protocols for secure computation that provide fairness and guaranteed output delivery. Consequently, many secure MPC protocols with these strong security properties

simply abort if a cheating is observed, realizing the weaker notion of *security with abort*. In particular, this means that, either the protocol succeeds and every party receives its outputs, or the protocol aborts, and this can happen even after the adversary has learnt the output of the computation, which could be a serious issue in some applications. One of the main drawbacks is that these protocols are vulnerable to *denial-of-service attacks* where corrupt parties can force the protocol to abort so that honest parties never learn the output of the computation.

This motivates the study of secure computation protocols with *identifiable abort* (ID-MPC) [BOS16, BOSS20, CL14, CFY17, IOZ14]. In this setting, if some malicious behaviour is detected or the adversary abort, the honest parties will agree upon the identity of at least one corrupt party. Even though this notion of security remains strictly weaker of fairness or guaranteed output delivery, it is very useful in practice as it discourages corrupt parties to behave maliciously, because upon abort at least one of them would be detected and maybe excluded from future computations.

1.1 Actively-secure MPC with dishonest majority

The last decade has seen a huge progress in the practicality of secure computations. Although it seems fairly natural to imagine efficient protocols with restricted security against semi-honest adversaries and/or assuming an honest majority, surprisingly a major advance has been in the dishonest majority case with active corruptions with the SPDZ line of works [DPSZ12, DO10, BDOZ11].

MPC in the correlated randomness model. A theoretically interesting and practically effective way to obtain efficiency in secure computation is by designing protocols with a randomness distribution phase, which is independent of the inputs to the function being computed, and sometimes also to the function itself. During this phase, parties receive randomness that are correlated from a pre-determined joint distribution. Using these random strings in the actual computation, it is possible to circumvent impossibility results such as impossibility of unconditional security in the plain model. Practically, one way to instantiate this model is through *MPC with pre-processing*.

Secure MPC protocols in this model restrict all the expensive operations to a pre-processing phase that can be both function and input independent. If this is the case we talk of *universal pre-processing* and if the pre-processing is only input independent, then we talk of *dedicated pre-processing*.

The randomness generated in the pre-processing stage is consumed by a lightweight non-cryptographic *online phase* that performs the actual circuit evaluation. Typically, the main goal of the pre-processing (or, *offline* phase) in MPC protocols is to produce randomness that enables an efficient, both in terms of communication and computation, evaluation of multiplication gates. In 1991, Beaver [Bea92] introduced a neat trick that permits efficient secure evaluation of circuits by randomizing the inputs to each multiplication gate using a pre-processed *random multiplication triple*. Using Beaver's trick, online evaluation turns out to be very efficient, involving only information-theoretic techniques, and creating triples

becomes the main bottleneck.

From passive to active security. Even though the pre-processing model allows to perform most of the work in a offline phase, leading to a very efficient online computation, this does not reduce the price to pay to have actively secure protocols compared to passively-secure ones. A typical example is the GMW protocol which needs expensive generic zero-knowledge (ZK) proofs to achieve active security. In 2008, Ishai, Prabhakaran and Sahai in [IPS08], described a novel technique, also known as IPS compiler, for actively secure MPC for Boolean circuits and constant number of parties, having asymptotic constant overhead over passively secure protocols. The IPS compiler is based on Oblivious Transfer (OT, see later for a proper definition of this important cryptographic primitive),and hence can also be expressed in the correlated randomness paradigm as OT can be pre-processed as shown by Beaver [Bea95]. In [LPO11], Lindell et al. presented a protocol based on the IPS compiler that converts semi-honest protocols in the dishonest majority setting into covertly secure ones. Later, Genkin et al. [GIP+14] proposed an MPC protocol for arbitrary number of parties based on Oblivious Linear Evaluation (OLE) for large fields with constant communication overhead. This technique was extended in [GIW16] to obtain active security for Boolean circuits. Recently, Hazay, Venkitasubramaniam and Weiss [HVW20], have proposed a more efficient compiler from passive to active that works over arbitrary fields and arbitrary number of parties. Almost all these works make black-box use of the underlying cryptographic primitives, OT, OLE, etc, and are mainly concerned with asymptotic complexity.

Concrete efficiency. A different line of works, more focused on *concrete* efficiency, started with the paper by Damgård and Orlandi [DO10] in 2010. To generate triples, the pre-processing phase utilizes an additively homomorphic encryption scheme, plus a "sacrifice" technique and homomorphic commitments to accomplish active security. This protocol uses commitments also during the circuit evaluation, still limiting the online phase to computational security. This issue was solved shortly after by Bendlin, Damgård, Orlandi and Zakarias [BDOZ11]. In this protocol, often called BDOZ, the homomorphic commitment scheme is replaced by a pairwise *information-theoretic* Message Authentication Code (MAC). A further optimization was introduced by Damgård, Pastro, Smart and Zakarias in 2012 with the SPDZ protocol [DPSZ12]. In SPDZ, the pairwise MAC used in BDOZ was simplified to a "global" MAC so that each party only stores a single field element for each MAC value instead of $n-1$ (where n is the number of parties). This leads to an information-theoretic online phase which is, roughly, only two times less efficient than the passive variant of the protocol. A second efficiency improvement provided by SPDZ is in the pre-processing phase, and comes from replacing the additive homomorphic encryption scheme with a somewhat homomorphic scheme. In particular, SPDZ uses the lattice-based scheme by Brakersy, Gentry and Vaikuntanathan (BGV) [BGV12], making extensive use of the packing technique of Smart and Vercauteren [SV14], which allows the manipulation of several plaintexts at once using SIMD (Single Instruction Multiple Data) operations.

After SPDZ, there has been a very large body of work mainly aiming to improve the SPDZ pre-processing phase, and in particular the triples generation step. In the next sections we will briefly describe some of these results, however we stress that what we are going to present is far from being exhaustive, and many interesting results are not going to be covered or even mentioned, due to space limitation. The aim of this paper is to introduce the main ideas and high level techniques used in protocols that are closely related to SPDZ, rather than giving a detailed and complete description of all the protocols dealing with active security and arbitrary number of corruptions.

1.2 Instantiating the preprocessing and alternative approaches

Concurrently to SPDZ, at CRYPTO 2012, another practical secure MPC protocol was presented by Nielsen et al. [NNOB12], usually referred to as TinyOT. It is a very efficient two-party protocol for secure computation of Boolean circuit, hence, in some sense, it can be considered complementary to SPDZ that on the contrary achieves better performances over large fields and allows arbitrary number of parties. TinyOT-online phase is very similar to SPDZ-online phase, except for the use of pair-wise MACs à la BDOZ. On the other hand, the offline phase differs significantly as it is based on oblivious transfer. TinyOT was later generalized to the multiparty case by Larraia et al. [LOS14], and to work on arithmetic circuit in MASCOT [KOS16]. The most efficient versions of the offline phase of SPDZ, TopGear [BCS19, KPR18], and MASCOT still represent the state-of-the-art of linear secret-shared based MPC for arithmetic and binary circuits, respectively. We provide a more detailed comparison between these two approaches in Section 5.

We mention here that in [KPR18] two different protocols were presented to improve SPDZ performances: LowGear and HighGear. The former uses an only-additive homomorphic encryption scheme and pairwise MACs, like BDOZ, but instantiates the encryption scheme with BGV, so to allow SIMD operations. Compared to BDOZ, LowGear does not perform ZK proofs of correct multiplications [BDOZ11], conjecturing that BGV satisfies the linear target malleability property described by Bitansky et al. [BCI+13]. LowGear is very efficient especially for a small number of parties due to the inherent packing it can use. HighGear is instead more similar to SPDZ, as it uses 1-leveled homomorphic BGV, and is more efficient for large number of parties.

Further improvements have been recently introduced by Hao Chen et al. [CKR+20]. This paper gives optimized ZK proofs by replacing BGV with BFV [Bra12, FV12]. By allowing the homomorphic encryption scheme to perform one more homomorphic operation, i.e. two instead of one, this protocol does not require the "sacrifice" technique to check triples correctness. Also, it generalizes Beaver's trick to generate "matrix triples" and "convolution triples".

Both the approaches described so far, either based on homomorphic encryption or oblivious transfer, use the pre-processing phase to produce multiplication triples as secret, correlated randomness to perform efficient circuit evaluation. However, it is possible to use different types of correlations, still achieving roughly the same online efficiency. For example we can have protocols based on oblivious linear-function evaluation (OLE) [NP99]

that can be seen as an arithmetic variant of oblivious transfer and used for arithmetic circuit [IPS09, ADI+17]; or protocols based on truth-tables correlations [CDv88, IKM+13, DNNR17, DKS+17, KOR+17, Cou19]. A formal description of these alternative approaches is out of the scope of this work, however we stress that some of these protocols achieve very good performances, in some applications better than protocols described in this survey. For example, using protocols based on truth-tables it is possible to obtain a better online efficiency at the price of a more expensive pre-processing.

Constant-round protocols. All the works mentioned above are based on a linear secret sharing scheme (LSSS) and gate-by-gate circuit evaluation. They require many rounds of communication (linear in the depth of the circuit), but very low bandwidth per gate. For this reason these protocols are very efficient in LAN (Local-Area-Network) setting, but not in WAN (Wide-Area-Network) setting. A different approach is taken by Yao (for the two-party case) and BMR (for the multiparty case) protocols. These mainly work for Boolean circuits (with few exceptions [AIK11, BMR16, Ben18, MW19]) and are based on the "garbled circuit" methodology. Roughly, in a first stage the parties generate an "encrypted" version of the circuit and inputs, and then, in a second stage, the circuit is evaluated without interactions. The number of communication rounds of these protocols is constant, they are usually slower that secret-sharing based protocols in LAN setting due to their higher bandwidth requirement and faster in WAN scenarios.

In the multiparty case, recent BMR-based protocols [LPSY15, LSS16, HSS17, WRK17b] achieve very good performances and have significantly narrowed the gap between secret-shared based and constant-round protocols. All these works are designed in the dedicated pre-processing model: in the input-independent phase a linear-secret sharing based protocol, typically multiparty TinyOT-like, is used to generate the garbled circuit, and in the online phase the parties evaluate the circuit locally except for one initial broadcast needed for the input step.

Mixed techniques. It is clear that these two flavours of MPC are mutually complementary, i.e. evaluation of some functions is more suited to the linear secret sharing paradigm, and for others it is more suited to the garbled circuit paradigm. For example, we would prefer to use Boolean circuits to perform computation over the integers where the operation is better expressed as a Boolean circuit, and arithmetic circuits where the computation to be performed is best expressed as an arithmetic circuit.
For this reason, that is to take the best of different methodologies and, at the same time, to mitigate their weakness, a different line of works tries to combine different secure computation paradigms. We mention some of them here. In [HKS+10, KSS13] we can find a method to convert between garbled circuits and additive homomorphic encryption in the semi-honest setting, in [DSZ15, RWT+18] it is described a general framework, for two-party computation, converting arithmetic sharing, Boolean sharing and garbled circuits with semi-honest security. This framework was generalized in [MR18] to the three-party case with active security. Recent works [RW19, AOR+19, EGK+20] show how to combine arithmetic

multiparty computation protocols, like SPDZ and BMR with active security.

1.3 Is MPC any good in practice? [Orl11]

Secure multiparty computation has been studied since the mid 1980s and back then the research on this field was mainly focused on feasibility results. Now, after almost 40 years, MPC is a rather mature technology that has rapidly progressed, especially in the last decade, from a notion of theoretical interest only into a technology that is starting to being commercialized.

In some aspects the extraordinary advance in the practicality of secure computation has been surprising, and it can be considered to be a consequence of a combination of algorithmic, technological and computational progress. Practically, if we consider malicious secure two party computation, the first implementation of Pinkas et al. [PSSW09] reports roughly 1114 sec for the evaluation of AES-128, i.e. a Boolean circuit of roughly 30000 gates (6400 AND gates and the rest XOR gates). Recent protocols [WRK17a, KOR+17] require roughly 10 ms for the same circuit. Possibly, recent improvements, for example in OT-extension protocols, will further improve these running times.

A number of online implementations are available, and SCALE-MAMBA [ACC+] and MP-SPDZ [Kel20] are the ones most closely related to SPDZ. We refer to [HHNZ19], for a more detailed discussion about these and other available frameworks. Even if we only focus on the dishonest majority setting, we want to remark that significant advances have also been made in the practicality of MPC protocols in different settings.

The current state of affairs is that secure MPC is able to compute relatively simple functions very efficiently, but it fails when we try to scale to more involved computations or involving a large number of parties. One of the main problem is communication, especially for linear secret-sharing (LSSS) based protocols like SPDZ or TinyOT. A major progress would consist in designing protocols that have both low bandwidth, like LSSS-based protocols, and small number of rounds, like GC-based protocols.

Despite the extraordinary progresses in the last years, there is still a long way to go before we can assert that secure multiparty computation is practically efficient in every scenario, for example for huge data sets or for Internet-like settings. Research in MPC is very active, and range from fundamental research, to implementation, to products deployments. It a very fast-moving field and, considering recent improvements, one would expect to see more breakthroughs in the area, with secure computation taking a leading role in most practical privacy-preserving solutions.

1.4 This survey

The aim of this work is to give an overview of the techniques used in concretely efficient MPC protocols with active security and dishonest majority, giving a high level description of the main building blocks of SPDZ and providing a limited literary review of the main related works. Other than describing the SPDZ protocol in its most recent and efficient version, we

also provide a rough description of the alternative approaches and, maybe more importantly, references to the relative papers.

We start with basic notation and preliminaries in Section 2. We explain how data is authenticated via information theoretic MACs in Section 3. Assuming a trusted functionality $\mathcal{F}_{\mathsf{Prep}}$ for the pre-processing phase, we describe the SPDZ online protocol in Section 4, and finally, in Section 5, we show how SPDZ implements $\mathcal{F}_{\mathsf{Prep}}$. In this section we also describe an alternative approach to the pre-processing implementation using oblivious transfer.

Acknowledgements. I would like to thank the organizers of WAIFI 2020 for inviting me to give a talk there. I am also grateful to Axel Mertens and Nigel Smart for helpful comments. This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT and by the FWO under an Odysseus project GOH9718N.

2 Preliminaries

We let κ (resp. s) denote the computational (resp. statistical) security parameter. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is negligible if for every positive polynomial[1] $p(\cdot)$, and all sufficiently large κ, it holds that $\mu(\kappa) < 1/p(\kappa)$. We use the abbreviation PPT to denote probabilistic polynomial-time. Let $\mathbb{F}$ denote a finite field, we consider protocols that allow to evaluate circuits C_f representing functions $f : \mathbb{F}^{n_{\mathtt{in}}} \to \mathbb{F}^{n_{\mathtt{out}}}$ with $n_{\mathtt{in}}$ inputs and $n_{\mathtt{out}}$ outputs. To ease the reading, we drop the dependence on f, when it is clear from the context.

We use lower case letters to denote finite field elements and bold lower case letters for vectors in $\mathbb{F}^\kappa$, for any finite field $\mathbb{F}$. If $\mathbf{x}, \mathbf{y}$ are vectors over $\mathbb{F}$, then $\mathbf{x} * \mathbf{y}$ denotes the component-wise products of the vectors. If A is a (probabilistic) algorithm then we denote by $a \leftarrow A$ the assignment of the output of A where the probability distribution is over the random tape of A and we denote by by $s \xleftarrow{\$} S$ the uniform sampling of s from a set S. We also use the notation $[d]$ as shorthand for the set of integers $\{1, \ldots, d\}$.

Security model. Protocols described in this paper work with n parties from the set $\mathcal{P} = \{P_1, \ldots, P_n\}$, and we consider security against malicious, static adversaries, i.e. corruption may only take place before the protocols start, corrupting up to $n - 1$ parties.

All the protocols described in this paper can be proved to be secure in the universal composition (UC) framework of Canetti [Can01]. Even though we omit these proofs here and provide the references to the relevant papers only, we will maintain some of the terminology used in the UC framework. For example we are going to use ideal functionalities in most of the protocols described in this survey. The reader who is not familiar with this notation and security model, and is not interested in understanding it, can simply imagine these functionalities as trusted entities that are called to securely perform some specific tasks.

Loosely speaking, protocols that aim to achieve security in the UC model are defined in three steps. First, the protocol and its execution in the presence of an adversary are

[1] Given a set S, a positive polynomial on S is such that $p(x) > 0$ for every $x \in S$

formalized, this represents the real-life model which we also call the *real world*. Next, an *ideal functionality* for executing the task is defined; its role is to act as a trusted party by separately receiving the input of each party, both honest and corrupt, and honestly computing the result of the protocol internally and returning the output assigned to each party. In this ideal process, also called *ideal world*, the parties do not communicate with one another but instead solely rely on the ideal functionality to provide them with their output. Finally, we say that the protocol in question UC-realizes the ideal functionality if running the protocol is equivalent, or *indistinguishable*, from emulating the ideal functionality. When we say that a protocol Π securely implements an ideal functionality $\mathcal{F}$ with computational (resp. statistical) security parameter κ (resp. λ), our theorems guarantee that the advantage in distinguishing the real and ideal executions is in $O(2^{-\kappa})$ (resp. $O(2^{-s})$).

Communication model. We assume all parties are connected via authenticated communication channels, as well as secure point-to-point channels and a broadcast channel. In practice, since we are considering security with abort, broadcast can be implemented with point-to-point channels requiring only two rounds of communication as follows [GL05]: 1) The party that needs to broadcast a value sends this to all parties; 2) All the receiving parties send the value they received to all other parties. It can be proven that either all the parties output the same value or the protocol aborts. This broadcast is also called *broadcast with abort*. It requires $O(n^2)$ communication per broadcast. SPDZ [DPSZ12] (Appendix A.3) describes how to optimize it in the case there are many broadcasts to perform, like in MPC protocols. Roughly, in all the broadcast instances parties maintain a running hash of all values sent and received, and these are checked later, at the end of the protocol. With this optimization the amortized cost per broadcast value is $O(n)$.

Secret sharing scheme. We only consider computation on values that are additively secret shared among parties, i.e. each shared value $x \in \mathbb{F}$ is represented as

$$\langle x \rangle = (x^{(1)}, \ldots, x^{(n)}),$$

where each party $P_i \in \mathcal{P}$ holds a random share $x^{(i)}$ and $x = \sum_{i \in [n]} x^{(i)}$. In this way, by setting all but a single share to be a random value in $\mathbb{F}$, we have that any subset of $n-1$ parties cannot recover the secret value x. We give a more formal definition below.

Definition 1 (Additive secret-sharing scheme) *Let $\mathbb{F}$ be a finite field and $n \in \mathbb{N}$ a positive integer. We define an additive secret sharing scheme $\mathbb{S} = (\mathsf{Share}, \mathsf{Recover})$ such that:*

- $\mathsf{Share}(x, n)$*: on input a secret x and an integer n, first it generates shares $(x^{(1)}, \ldots, x^{(n-1)})$ uniformly at random from $\mathbb{F}$ and define $x^{(n)} = x - \sum_{i=1}^{n-1} x^{(i)}$; then it outputs $(x^{(1)}, \ldots, x^{(n)})$, where $x^{(i)}$ is the share of party P_i*

- $\mathsf{Recover}(\mathsf{x}^{(1)}, \ldots, \mathsf{x}^{(n)})$*: given all the shares $x^{(i)}, i \in [n]$, parties compute $x = \sum_{i=1}^{n} x^{(i)}$.*

Trivially, this secret sharing scheme is linear, therefore linear operations can be performed locally without interactions among parties, as described below.

- Addition of secret-shared values: $\langle x \rangle + \langle y \rangle = (x^{(1)} + y^{(1)}, \ldots, x^{(n)} + y^{(n)}) = \langle x + y \rangle$

- Addition by a public value a: $a + \langle x \rangle = (a + x^{(1)}, \ldots, x^{(n)}) = \langle a + x \rangle$

- Multiplication by a public value a: $a \cdot \langle x \rangle = (a \cdot x^{(1)}, \ldots, a \cdot x^{(n)}) = \langle a \cdot x \rangle$

Statistical distance. Let E be a finite set, Ω be a probability space and $X, Y : \Omega \to E$ be random variables. The *statistical distance* between X, Y is defined as:

$$\Delta(X, Y) = \frac{1}{2} \sum_{x \in E} \left| \Pr_X(X = x) - \Pr_Y(Y = x) \right|$$

We recall the following result from [AJL+12].

Lemma 1 (Smudging Lemma) *Let B_1 and B_2 be positive integers, let $e \in [-B_1, B_2]$ be a fixed integer, and let E_1, E_2 be independent random variables uniformly distributed in $[-B_1, B_2]$. Define the two stochastic variables $X_1 = E_1 + e$ and $X_2 = E_2$. Then, it holds that:*

$$\Delta(E_1, E_2) < B_1 / B_2.$$

This lemma allows to "smudge out" small differences between distributions adding large noise. It will be used many times in the protocols we describe in the next sections, often implicitly.

2.1 Threshold (L-leveled) homomorphic encryption

We briefly recall the definition of threshold (L-leveled) homomorphic encryption (THE) [AJL+12, BGG+18] scheme. It is similar to a standard (leveled) homomorphic encryption scheme, but with different key-generation and decryption algorithms. The scheme is parametrized by security parameters (κ, s), the number of levels L, the amount of packing of plaintext elements which can be made into a single ciphertext N, and by a linear secret scheme $\mathbb{S}$. We instantiate $\mathbb{S}$ with an additive secret sharing scheme as describe before, and hence we give a less general definition of a THE scheme. Informally, a threshold L-leveled HE scheme supports homomorphic evaluation of any circuit C consisting of addition and multiplication gates and of multiplicative depth at most L, with the provision for distributed (threshold) decryption.

Definition 2 *An L-leveled public key homomorphic encryption scheme with message space $\mathcal{M} = \mathbb{F}^N$, is a tuple of PPT algorithms $\mathsf{THE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{PartDec}, \mathsf{DistDec})$, satisfying the following specifications:*

$(\mathsf{pk}, \mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(n)}) \leftarrow \mathsf{KeyGen}(1^\kappa, 1^s, n)$: *taking as input the security parameters κ and s, and the number of parties n, it outputs a public key* pk*, and secret key additive shares* $(\mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(n)})$*;*

$\mathfrak{ct} \leftarrow \mathsf{Enc}(m; \mathsf{pk})$: *it takes a plaintext $m \in \mathcal{M}$ and public key* pk*, and output a ciphertext* $\mathfrak{ct}$*;*

$\hat{\mathfrak{ct}} \leftarrow \mathsf{Eval}(C, \mathfrak{ct}_1, \ldots, \mathfrak{ct}_t)$: *it takes as input a circuit $C : \mathbb{F}^t \to \mathbb{F}$, with multiplicative depth at most L and t ciphertexts $\mathfrak{ct}_1, \ldots, \mathfrak{ct}_t$, and outputs an evaluation ciphertext $\hat{\mathfrak{ct}}$;*

$(p^{(1)}, \ldots, p^{(n)}) \leftarrow \mathsf{PartDec}(\mathfrak{ct}; \mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(n)}$: *given a ciphertext $\mathfrak{ct}$ and a secret key share $\mathfrak{ct}^{(i)}$, it outputs a partial decryption $p^{(i)}$ to party P_i, for every $i \in [n]$;*

$\hat{m} \leftarrow \mathsf{DistDec}(p^{(1)}, \ldots, p^{(n)}; \mathsf{pk})$: *it takes as input the public key and all the partial decryption outputs, and outputs a plaintext $\hat{m}$.*

The scheme needs to satisfy correctness, semantic security and simulation security as described in [BGG+18]. Here we omit the formal definitions of these properties. While the first two are relatively standard, the latter essentially says that no information about the key shares and plaintext should be leaked by the decryption algorithms other than what is already implied by the result of homomorphic operations.

In SPDZ, a THE scheme is used to generate random triples in the pre-processing phase, therefore we need a very simple THE supporting only one homomorphic operation, i.e. L=1. Concretely, the THE scheme is instantiated with the scheme by Brakersky, Gentry and Vaikuntanathan (BGV) [BGV12], based on the Ring Learning with Error assumption [Reg05, LPR13], and supporting packing operations [SV14] that permits to handle many plaintexts in a single ciphertext. We omit the description of BGV in this survey.

2.2 UC commitments

Functionality $\mathcal{F}_{\mathsf{Commit}}$

Commit: On input $(\texttt{Commit}, m, i, \tau_m)$ from P_i, store (m, i, τ_m). τ_m is a handle for the commitment. and output (i, τ_m) to all parties.

Open: On input $(\mathsf{Open}, i, \tau_m)$ by P_i, output (m, i, τ_m) to all parties.

If instead $(\texttt{NoOpen}, i, \tau_m)$ is given by the adversary, and P_i is corrupt, the functionality outputs $(\perp, i, \tau_m)$ to all parties.

Figure 1: Commitments functionality.

A commitment scheme allows a commiter holding a secret value m to send a commitment c of m to a verifier, and later on to "open" this commitment to reveal m. More formally, a commitment scheme is defined by three algorithms.

- $\mathsf{Setup}(1^\kappa)$: given as input the security parameter, it generates the global parameters that will be implicitly used by the other algorithms;

- $(c, w) \leftarrow \mathsf{Commit}(m)$: given a message m it produces a commitment c on m and the opening information w;

- $m \leftarrow \mathsf{Open}(c, w)$: it decommits c using w and outputs either the message m or $\perp$ if the opening fails.

The scheme has to be both *binding*, i.e. the opening should successfully open to one value only, and *hiding* which means that the commitment c should not reveal any information about m. These two properties can be achieved in a perfect, statistical or computational way. A UC-secure commitment must be both *extractable* (meaning that it is possible to extract the value that a corrupted party commits to) and *equivocable* (meaning that it is possible to generate commitments that can be opened to any value). In this survey we will use an ideal functionality $\mathcal{F}_{\mathsf{Commit}}$ as described in Figure 1.

This ideal functionality can be implemented assuming a random oracle, by defining $c = \mathsf{H}(m, i, r)$, where H is a random oracle, $r \leftarrow \{0,1\}^\kappa$ and $w = (c, r)$.

Using this hash-based commitment we can also efficiently implement a standard coin flipping functionality $\mathcal{F}_{\mathsf{Rand}}$. We refer to [DKL+13] for more details.

2.3 Zero knowledge proofs

A zero-knowledge (ZK) proof [GMR89] is an interactive protocol between a prover P and a verifier V that allows the prover to demonstrate that a statement is true without revealing any further information about the proof beyond the fact that the statement is true.
An NP-relation $\mathcal{R}(x, w)$ is an efficiently decidable binary relation $\mathcal{R}(x, w)$ that is polynomially bounded, i.e. if $\mathcal{R}(x, w)$ is satisfied, then $|w| \leq \mathsf{poly}(|w|)$. Any NP-relation defines a language $\mathcal{L} = \{x : \exists w, \mathcal{R}(x, w) = 1\}$. Usually w is called a *witness* for the statement $x \in \mathcal{L}$.

A ZK proof protocol for the NP relation $\mathcal{R}(x, w)$, with common input x and additional input w for P, satisfies three properties that we can informally describe as follows:

- Completeness: if $x \in \mathcal{L}$, and P knows a proof of this, she/he will succeed in convincing V;

- Soundness: : if the statement is false, no prover can convince the verifier of the truth of the statement except with probability ϵ, where ϵ is the *soundness error* of the protocol;

- Zero-knowledge: the interaction between P and V yields nothing beyond the fact that the statement is true. This is equivalent to require the existence of a simulator that can produce an honest-looking transcript for the protocol, without knowing anything about the statement.

Functionality $\mathcal{F}_{\mathsf{OT}}^k$

Running between a sender P_S and a receiver P_R, it operates as follows.

- P_S inputs $(x_0, x_1) \in \{0,1\}^k \times \{0,1\}^k$ and P_R inputs b.
- The functionality outputs x_b to P_R.

Figure 2: Functionality for one-out-of-two oblivious transfers on k-bit strings.

2.4 Oblivious transfer

Oblivious transfer is a fundamental cryptographic primitive originally introduced by Rabin [Rab81] and Wiesner [Wie83]. Subsequent works by [Kil88, EGL85] showed oblivious transfer to be a very powerful primitive. In particular, Kilian [Kil88] showed that OT is complete for secure multi-party computation. Many MPC protocols have been constructed based on OT, including the GMW protocol, Yao's garbled circuits and the IPS compiler that we have already mentioned before.

In its classical formulation, a (one-out-of-two) oblivious transfer is a two-party protocol between a sender P_S and a receiver P_R: the sender inputs two messages x_0, x_1, a receiver inputs a bit b, and the goal is for the receiver to learn x_b and nothing more, whilst the sender learns no information about b. In Figure 2 we describe the ideal functionality for oblivious transfer on bit strings of length k, meaning that sender's inputs x_0, x_1 are elements in $\{0,1\}^k$. Given the inputs from P_S and P_R the functionality outputs the string x_b corresponding to receiver's input b.

Oblivious transfer extension. Although oblivious transfer is a fundamental building block for many cryptographic constructions, it used to be considered an expensive primitive. Indeed, Impagliazzo and Rudich [IR89] showed a black-box separation result that is strong evidence that OT is impossible without the use of expensive public-key cryptography. However, thanks to recent, and somehow surprisingly, advances in the field, we can fairly claim that in practice OT is no longer an expensive primitive.

Beaver in [Bea96] first showed that OT can be "extended", i.e. starting from few OTs one could generate a large amount of additional OTs using only cheap symmetric primitives. Albeit elegant, Beaver's protocol is highly impractical. The first efficient OT-extension protocol was described by Ishai, Kilian, Nissim and Petrank [IKNP03] in the passive setting. Subsequent works, secure against both passive [ALSZ13, KK13] and active [ALSZ15, KOS15, OOS17] adversary, all follow the IKNP blueprint. These protocols are computationally very efficient and allow to create more than 10 million of OTs in 1 sec. The main bottleneck remains communication. A different approach, that uses LPN-based (LPN stands for Learning Parity with Noise[2]) PCG (i.e. Pseudorandom Correlation Generators see Section 5.3) for OT-extension, outperforms previous solutions in terms of communications in

[2]Roughly, the LPN assumption says that given a random linear code C, a noisy random codeword of C is pseudo-random.

low bandwidth network, but at price of high computational overhead [BCG+19a, SGRR19]. A very recent protocol by Yang et al. [YWL+20] achieves impressive performances both in terms of communication and computation requiring only 21 nanoseconds (resp. 22 nanoseconds) for generating one (correlated) OT in a 50 Mbps network with passive (resp. active) security. Correlated OT (COT) is a slightly different variant of OT, but sufficient for many practical MPC protocols. We will define COT in Section 5.2.

3 Data representation

While an additive secret sharing scheme is sufficient to guarantee privacy and hence security in the weak model of semi-honest security, we need extra caution in presence of an active adversary in order to prevent corrupt parties to inject incorrect values to the protocol that could lead to erroneous results or information leakage.

As we mentioned in the introduction, SPDZ protocols achieve active security by authenticating each shared value with an information-theoretic MAC. This can be done either in a pairwise manner [BDOZ11, NNOB12], or in a global manner [DPSZ12, LOS14]. Both of these variants can be applied, yet implying significant practical differences in the total amount of data each party needs to store, in the ZK proofs and in the way MACs are checked. We describe both these variants below.

BDOZ-style MAC: Each value $x \in \mathbb{F}$ is authenticated and additively secret shared among parties in $\mathcal{P}$ in such a way that each party P_i holds a share $x^{(i)}$ and $n-1$ pairwise MACs

$$\mathsf{m}_x^{(ij)} = \mathsf{k}_x^{(ji)} + x^{(i)} \cdot \Delta^{(j)},$$

for each $j \neq i$. This notation implies that P_i holds $x^{(i)}$ and $\{\mathsf{m}_x^{(ij)}\}_j$, and each other party P_j holds a local key $\mathsf{k}_x^{(ji)}$, i.e. depending on the value $x^{(i)}$, and a global key $\Delta^{(j)}$ fixed for the entire computation. The values $\mathsf{m}_x^{(ij)}, \mathsf{k}_x^{(ji)}, \Delta^{(j)}$ are either elements of $\mathbb{F}$, or elements of an extension field $\mathbb{E}$ of $\mathbb{F}$. Typically, if $\log_2 |\mathbb{F}| \geq \kappa$, then $\mathbb{E} = \mathbb{F}$. We will use the following notation to represent this type of authenticated values:

$$[x]_B^j = \big(\langle x \rangle, (\mathsf{m}_x^{(1j)}, \ldots, \mathsf{m}_x^{(nj)}), (\mathsf{k}_x^{(j1)}, \ldots, \mathsf{k}_x^{(jn)}), \Delta^j\big),$$

to denote each party authenticating their share of $\langle x \rangle$ towards party P_j, and

$$[x]_B = \big([x]_B^1, \ldots, [x]_B^n\big),$$

for the global representation. It is easy to see that parties can locally perform linear operations on authenticated data.

Addition of pairwise authenticated secret-shared values.

$$[x]_B + [y]_B \ = \big([x]_B^1 + [y]_B^1, \ldots, [x]_B^n + [y]_B^n\big) = \big([x+y]_B^1, \ldots, [x+y]_B^n\big)$$

$$= [x+y]_B,$$

since, for each i, it holds:

$$\begin{aligned}[x]_B^i + [y]_B^i &= \big(\langle x\rangle + \langle y\rangle, (\mathsf{m}_x^{(1j)} + \mathsf{m}_y^{(1j)}, \ldots, \mathsf{m}_x^{(nj)} + \mathsf{m}_y^{(nj)}), (\mathsf{k}_x^{(j1)} + \mathsf{k}_y^{(j1)}, \ldots, \mathsf{k}_x^{(jn)} + \mathsf{k}_y^{(jn)}), \Delta^j\big) \\ &= \big(\langle x+y\rangle, (\mathsf{m}_{x+y}^{(1j)}, \ldots, \mathsf{m}_{x+y}^{(nj)}), (\mathsf{k}_{x+y}^{(j1)}, \ldots, \mathsf{k}_{x+y}^{(jn)}), \Delta^j\big)\end{aligned}$$

Addition by a public value. Given a publicly known value $a \in \mathbb{F}$,

$$a + [x]_B = [a+x]_B,$$

where $\langle a+x\rangle$ is obtained as described in the introduction. All the MAC values and keys remain the same, except for $\mathsf{k}_{a+x}^{(j1)} = \mathsf{k}_x^{(j1)} - a \cdot \Delta^{(j)}$.

Multiplication by a public value. As before, given a public $a \in \mathbb{F}$, $a \cdot [x]_B = [a \cdot x]_B$, obtained by multiplying each share, MAC and local key by a.

SPDZ-style MAC: Each value $x \in \mathbb{F}$ is additively secret shared and authenticated as follows.

$$[x]_S = (\langle x\rangle, \langle \mathsf{m}_x\rangle, \langle \Delta\rangle),$$

where $\mathsf{m}_x = \sum_i \mathsf{m}_x^{(i)} = x \cdot \Delta$ and $\Delta = \sum_i \Delta^{(i)}$ is the MAC key, that is hence unknown to the parties. As for $[\cdot]_B$, we assume m_x and Δ to be elements of $\mathbb{F}$ or $\mathbb{E}$, such that $\mathbb{F} \subset \mathbb{E}$. Again, due to the linear relation between authenticated values and MAC, linear operations can be carried out locally.

Addition of pairwise authenticated secret-shared values. $[x]_S + [y]_S = (\langle x\rangle + \langle y\rangle, \langle \mathsf{m}_x\rangle + \langle \mathsf{m}_y\rangle, \langle \Delta\rangle) = (\langle x+y\rangle, \langle \mathsf{m}_x + \mathsf{m}_y\rangle, \langle \Delta\rangle) = [x+y]_S$.

In a similar way we can perform addition and multiplication by a public value. Note that given a public value a, the MAC value on a is defined by each party setting $\mathsf{m}_a^{(i)} = a \cdot \Delta^{(i)}$, to obtain a valid authenticated share.

Conversion to $[\cdot]_S$. It is possible to locally convert the BDOZ representation to the SPDZ representation [LOS14]. As we will see in Section 5, this conversion is particularly useful in the case we want to use a two-party primitive, like oblivious transfer, to generate authenticated values and random triples. This allows a more efficient memory usage and exploits a less expensive, global MAC check procedure in the online evaluation. Given a BDOZ-style authenticated value $[x]_B$, parties already hold $\langle x\rangle$ and additive shares of Δ, so to obtain a SPDZ-style representation, it is enough to generate shares

$\mathsf{m}_x^{(i)}$ of $\mathsf{m}_x = x \cdot \Delta$. This is done without any interaction by parties combining their pairwise MACs and keys as follows:

$$\mathsf{m}_x^{(i)} = \sum_{j \neq i} \left(\mathsf{m}_x^{(ij)} - \mathsf{k}_x^{(ij)} \right) + x^{(i)} \cdot \Delta^{(i)}.$$

Indeed the following relations hold:

$$\begin{aligned} \mathsf{m}_x = \sum_i \mathsf{m}_x^{(i)} &= \sum_i x^{(i)} \cdot \Delta^{(i)} + \sum_i \sum_{j \neq i} \left(\mathsf{m}_x^{(ij)} - \mathsf{k}_x^{(ij)} \right) \\ &= \sum_i x^{(i)} \cdot \Delta^{(i)} + \sum_i \sum_{j \neq i} x^{(i)} \cdot \Delta^{(j)} \\ &= x \cdot \Delta. \end{aligned}$$

3.1 Checking MACs

During the online evaluation of the circuit, parties need to communicate or, more precisely, they need to be able to *reveal* secret shared values $[x]$. This is done by sending over all the private shares $x^{(i)}$ of $\langle x \rangle$:

Open: On input (Open, x) from every party, each P_i broadcasts $x^{(i)}$, recovers $x = \sum_i x^{(i)}$ and stores x.

Moreover, we need to prevent corrupt parties to disclose incorrect values, therefore we need a way to check MACs on opened values. This can be done in different ways. Note that since it is always possible to convert from $[\cdot]_B$ to $[\cdot]_S$, and the latter allows a more efficient check, we only consider the case of SPDZ-style MAC.

An obvious way to check if a reconstructed value is correct is by revealing shares $\mathsf{m}_x^{(i)}$ and $\Delta^{(i)}$, for each $i \in [n]$, along with $x^{(i)}$. Clearly, we can perform this check only once because after the MAC key Δ is revealed all parties can forge new MACs and introduce incorrect values, so a new MAC key should be generated (along with new pre-processed material).

To overcome this problem in original SPDZ protocol, parties wait until the end of the computation to reveal the MACs and the MAC key. Only when the circuit evaluation is completed, parties check the MACs on opened values, and if the check passes the final result of the computation is opened.

However, this approach limits the use of Δ to a single evaluation, preventing reactive computations without generating fresh MAC keys and pre-processed randomness. The following two procedures, firstly described in [DKL+13], allow to check a single MAC and a batch of MACs, respectively, on opened values *without disclosing the global MAC key*. At a high level, given an opened value $\tilde{x}$ and authenticated value $[x]$, the goal is to check whether $\tilde{x} = x$ by checking the MAC relation $\mathsf{m}_x = \tilde{x} \cdot \Delta$. To this end parties broadcast $\sigma^{(i)} = \mathsf{m}_x^{(i)} - \tilde{x} \cdot \Delta^{(i)}$ and then check that $\sum_i \sigma^{(i)} = 0$. Note that the shares $\mathsf{m}_x^{(i)}$ are uniformly random, so sending $\sigma^{(i)}$ does not leak any private information, in particular about $\Delta^{(i)}$.

MAC Check. On input $(\mathsf{CheckMAC}, \tilde{x})$ from all parties:

1. Each party P_i computes $\sigma^{(i)} = \mathsf{m}_x^{(i)} - \tilde{x} \cdot \Delta^{(i)}$
2. P_i calls the functionality $\mathcal{F}_{\mathsf{Commit}}$ on command $(\mathtt{Commit}, \sigma^{(i)}, i, \tau^{(i)})$ to broadcast $(i, \tau^{(i)})$
3. All parties call $\mathcal{F}_{\mathsf{Commit}}$ with command $(\mathsf{Open}, i, \tau^{(i)})$, obtaining $\sigma^{(i)}$, for all $i \in [n]$
4. Parties check if $\sigma^{(1)} + \cdots + \sigma^{(n)} = 0$. If the check passes, accept x as a correct authenticated value, otherwise output $\perp$ and abort.

Batch MAC Check. On input $(\mathsf{CheckMAC}, \tilde{x}_1, \ldots \tilde{x}_t)$ from all parties:

1. Parties use $\mathcal{F}_{\mathsf{Rand}}$ to sample a random vector $\mathbf{r} \leftarrow \mathbb{F}^t$
2. Each party locally computes $\tilde{x} = \sum_{j=1}^{t} r_j \cdot \tilde{x}_j$
3. Each party P_i computes $\langle \tilde{\mathsf{m}} \rangle \leftarrow \sum_{i=1}^{t} r_j \cdot \langle \mathsf{m}_{\tilde{x}_j} \rangle$ and $\langle \sigma \rangle = \langle m \rangle - \tilde{x} \cdot \langle \Delta \rangle$
4. Use the (single) **MAC Check** procedure described above to check the MAC relation on the value $\tilde{x}$, with MAC $\tilde{\mathsf{m}}$ and Δ.

Remark 1 *These procedures use the $\mathcal{F}_{\mathsf{Commit}}$ (Figure 1) functionality so that a corrupt party is not able to cheat in the broadcast of its share of σ, for example using information on shares sent by honest parties. In the same spirit, during the* **Batch MAC Check***, the sampling of the vector $\mathbf{r}$, used to generate random linear combinations of opened values and MACs, is performed by $\mathcal{F}_{\mathsf{Rand}}$ to ensure that corrupt parties are not able to influence it in the attempt of passing the check with incorrect shares.*

We omit the security proof of the MAC Check procedure, however, the intuition is that if corrupt parties send incorrect shares, such that the opened value is $x+\delta$, for some adversarial chosen δ, then to pass the check it should hold that

$$\sum_{i \in [n]} \sigma^{(i)} = (x + \delta) \cdot \Delta - \sum_{i} \mathsf{m}_x^{(i)} = (x + \delta) \cdot \Delta - \mathsf{m}_x = 0.$$

This means that the adversary should be able to "correct" the corrupt parties' MAC share by the value $\delta \cdot \Delta$, which in turns implies to guess the global key Δ. Hence the probability of passing the check is $1/|\mathbb{E}|$, which is negligible when the field is large. As a consequence, to carry out computation over small fields we need to take a large enough extension field $\mathbb{E}$ and embed the whole computation in that field, generating a significant communication overhead.

Another issue with this technique is that it does not work over other rings, for example over the modular rings $\mathbb{Z}/2^k\mathbb{Z}$. The reason is that these rings contain zero divisors and hence guessing $\delta \cdot \Delta \mod 2^k$ is much easier than over fields. We discuss in the next sections how to overcome these difficulties.

3.2 Mini MAC

In 2013, Damgård and Zakarias [DZ13] introduced a new technique, called MiniMAC, that helps to reduce the overhead in case of computation over small fields. The core idea of MiniMAC is that of performing batch computation, for example evaluating in parallel several instances of the same circuit over small fields, or performing single computation of a "well-formed" circuit. More concretely, a "well formed" circuit is a circuit such that every layer contains a large enough number of gates and its outputs are input of the next layer, so to allow a number of parallel computation at time for each layer. This requires, most of the time, to "pre-process" the circuit to be evaluated to obtain a circuit of the desired form, slightly increasing the complexity of the protocol. The main difference between MiniMAC and SPDZ is in the way values are authenticated.

MiniMAC authentication

Let C be an $[N, k, d]$ linear error correcting code over $\mathbb{F}$ of length N, dimension k and Hamming distance d. We recall that the Hamming distance of a linear code is the smallest Hamming distance between any two different codewords, and is equal to the minimum Hamming weight of the non-zero codewords in the code. Let $\mathsf{Encode} : \mathbb{F}^k \to \mathbb{F}^N$ be a systematic encoding algorithm that maps a vector $\mathbf{x} = (x_1, \cdots, x_k)$ into a codeword $\mathbf{c_x} \in C$, such that the first k entries of $\mathbf{c_x}$ are equal to $\mathbf{x}$, i.e. $\pi_k(\mathbf{c_x}) = \mathbf{x}$, where π_k denotes the projection map $\pi_k : \mathbb{F}^N \to \mathbb{F}^k$, with $\pi_k(c_{x,1}, \ldots, c_{x,N}) = (c_{x,1}, \ldots, c_{x,k})$. Given the code C, the Shur-transform of C, denoted by C^*, is a linear $[N, k^*, d^*]$ code defined as the span of the set $\{\mathbf{c_x} * \mathbf{c_y} | \mathbf{c_x}, \mathbf{c_y} \in C\}$. It can be shown that $k^* \geq k$ and $d^* \leq d$. Let $\mathbf{x} \in \mathbb{F}^k$, we define $C^*_{\mathbf{x}} = \{\mathbf{c}^* \in C^* | \pi_k(\mathbf{c}^*) = \mathbf{x}\}$, i.e. the set of codewords in C^* such that $\mathbf{x}$ appears in the first k coordinates.

Remark 2 *Since $\mathbf{c_x} * \mathbf{c_y} \in C^*$, and $\pi_k(\mathbf{c}_x * \mathbf{c}_y) = \mathbf{x} * \mathbf{y}$, because C is systematic, it holds that:*

$$\mathbf{c_x} * \mathbf{c_y} \in C^*_{\mathbf{x}*\mathbf{y}}.$$

We are now ready to introduce the MiniMAC authentication technique. Roughly, given vector $\mathbf{x} = (x_1, \ldots, x_k) \in \mathbb{F}^k$ and an error correcting code C as described above, we define the MAC on $\mathbf{x}$ by first encoding $\mathbf{x}$ using Encode to obtain $\mathbf{c_x}$ and then setting $\mathbf{m_x} = \mathbf{c_x} * \boldsymbol{\Delta}$, with $\boldsymbol{\Delta} \in \mathbb{F}^N$.

MiniMAC-style MAC: Given an error correcting code C as described before, and a vector $\mathbf{x} = (x_1, \ldots, x_k)$ additively secret shared, we represent MiniMAC authentication as follows:

$$[\mathbf{x}]_M = \big(\langle \mathbf{x} \rangle, \langle \mathbf{c_x} \rangle, \langle \mathbf{m_x} \rangle, \langle \boldsymbol{\Delta} \rangle\big),$$

with $\mathbf{x} = \sum_i \mathbf{x}^{(i)} = \sum_i (x_1^{(i)}, \ldots, x_k^{(i)})$, $\mathbf{m_x} = \sum_i \mathbf{m}_{\mathbf{x}}^{(i)} = \mathbf{c_x} * \boldsymbol{\Delta}$. We also need another representation $[\cdot]^*_M$ that is similar to $[\cdot]_M$, except that uses C^* instead of C. This second representation in particular is needed to multiply a $[\cdot]_M$-representation by a public constant, as shown below.

Addition of authenticated values. This is a straightforward generalization to vectors of addition of two $[\cdot]_S$-representations.
Addition by a public value. Given a public vector $\mathbf{a}$,

$$\mathbf{a} + [\mathbf{x}]_M = \big((\mathbf{a} + \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}), \mathbf{c_a} + \mathbf{c_x}, \\ (\mathbf{m}_\mathbf{x}^{(1)} - \mathbf{c_a} * \boldsymbol{\Delta}^{(1)}, \ldots, \mathbf{m}_\mathbf{x}^{(n)} - \mathbf{c_a} * \boldsymbol{\Delta}^{(n)}) = [\mathbf{a} + \mathbf{y}]_M$$

Multiplication by a public value. For this we need both $[\cdot]_M$ and $[\cdot]^*_M$

$$\mathbf{a} * [\mathbf{x}]_M = \Big(\big((\mathsf{Encode}^*)^{-1}(\mathbf{c_a} * \mathbf{c}_\mathbf{x}^{(1)}), \ldots, (\mathsf{Encode}^*)^{-1}(\mathbf{c_a} * \mathbf{c}_\mathbf{x}^{(n)}) \big) \\ (\mathbf{c_a} * \mathbf{m}_x^{(1)}, \ldots, \mathbf{c_a} * \mathbf{m}_\mathbf{x}^{(n)}) \Big) = [\mathbf{s}]^*_M,$$

where $\pi_k(\mathbf{s}) = \mathbf{a} * \mathbf{x}$ and $(\mathsf{Encode}^*)^{-1}(\mathbf{c}^*)$ is the vector in $\mathbb{F}_2^{k^*}$ corresponding to the codeword $\mathbf{c}^*$ in C^*.

Hence, when we perform multiplication by a known value we produce a $[\cdot]^*_M$-sharing. Note that we cannot perform multiplication with $[\cdot]^*_M$, and we need to convert $[\cdot]^*_M$ back to $[\cdot]_M$ after each multiplication.

Conversion from $[\cdot]^*_M$ to $[\cdot]_M$. We need a double encoded value $([\mathbf{r}]_M, [\mathbf{r}]^*_M)$. Let $[\mathbf{x}]^*_M$ the value we need to convert, parties do the following.

1. Open $[\mathbf{x}]^*_M - [\mathbf{r}]^*_M$ obtaining $\mathbf{c}^*_{\mathbf{x}-\mathbf{r}} \in C^*_{\mathbf{x}-\mathbf{r}}$ and from the first k coordinates of this value get $\mathbf{x} - \mathbf{r}$
2. P_1 computes $\mathsf{Encode}(\mathbf{x} - \mathbf{r}) = \mathbf{c}_{\mathbf{x}-\mathbf{r}}$ and broadcast it
3. Parties check that $\mathbf{c}^*_{\mathbf{x}-\mathbf{r}}$ and $\mathbf{c}_{\mathbf{x}-\mathbf{r}}$ are valid codewords encoding the same value and compute $[\mathbf{x}]_M = (\mathbf{x} - \mathbf{r}) + [\mathbf{r}]_M$.

We do not describe here the MAC Check for the MiniMAC representation, since it is very similar to the one described for SPDZ-style authentication. Loosely speaking, it is essentially a batch check through a random linear combination of codewords corresponding to the opened values. Intuitively, in this case an adversary introducing an incorrect value would pass the check if it correctly guesses at least d coordinates of the global key, where d is the Hamming distance of the code C. In this way, the probability of successfully cheating is given by $|\mathbb{F}|^{-d}$, which is negligible when d is big enough.

3.2.1 Related and subsequent works

The original MiniMAC paper [DZ13], defines both the MAC scheme we have just described and the corresponding online protocol that uses this representation. In a follow-up work Damgård, Lauritsen and Toft [DLT14], introduced several optimizations and the first implementation of MiniMAC, reporting only 4ms to evaluate AES-128 on a Boolean circuit,

without pre-processing. The first dedicated construction of MiniMAC multiplication triples and other pre-processed material was given by Frederiksen et al. [FKOS15] using oblivious transfer.

A similar approach was also followed by the Committed MPC protocol [FPY18] of Frederiksen, Pinkas and Yanai, where the information-theoretic MAC just described is replaced by UC-secure homomorphic commitments based on error-correcting codes. Another very promising work by Cascudo and Gundersen [CG20] has recently appeared. Again it follows the same core idea of MiniMAC, but it uses the algebraic notion of *reverse multiplication friendly embedding* (RMFE) [CCXY18] to encode values instead of linear codes. The main advantage of this approach is that it allows to use smaller fields and hence achieve better communication complexity, considering also the pre-processing phase.

3.3 SPDZ over $\mathbb{Z}/2^k\mathbb{Z}$

The MAC scheme described in the previous section only works over fields and not over rings, where guessing $\delta \cdot \Delta$, for some adversarial chosen δ, would be much easier than guessing Δ. In [CDE+18], Cramer et al. present a new authentication scheme, closely related to the ones described above, that allows secure active computation over the ring $\mathbb{Z}/2^k\mathbb{Z}$. Secure computation over this ring is useful in many applications, and could significantly simplify implementations, such as in the case of evaluations of functions containing comparisons and bit-wise operations.

Roughly, the protocol described in [CDE+18], called SPD$\mathbb{Z}_{2^k}$, achieves security running over a larger ring modulo 2^{k+s} instead of 2^k, where s is approximately the statistical security parameter, even if correctness is only guaranteed modulo 2^k. SPD$\mathbb{Z}_{2^k}$ pre-processing is based on oblivious transfer using a MASCOT-like approach, and it is particularly efficient in a LAN setting [DEF+19]. A different approach is taken in [OSV20], where the SPD$\mathbb{Z}_{2^k}$ pre-processing is implemented using a "$\mathbb{Z}_{2^k}$-friendly" version of BGV, hence more similarly to SPDZ and Overdrive, yielding to a protocol more suited for a WAN scenario and larger number of parties. In [CDFG20], Catalano et al. describe another implemetation of the pre-processing phase using the Joye-Libert [JL13] additively homomorphic encryption crytposystem and pairwise BDOZ-style authentication. This protocol is designed for the two-party case and is more efficient for large choices of k.

4 SPDZ online evaluation

In this section we describe the circuit evaluation phase (or, online phase) of SPDZ, assuming a trusted setup, $\mathcal{F}_{\mathsf{Prep}}$, that generates the correlated randomness used in the actual circuit computation. Recall that we assume that the circuit being evaluated is an arithmetic circuit over the finite field $\mathbb{F}$.

The main question is "What do we need for $\mathcal{F}_{\mathsf{Prep}}$?" The minimal requirements are the following. 1) Random authenticated values, $(r, [r]_S)$, that are used as masks to create authenticated sharings of the inputs. The value r is secret shared, but known to the input

party; 2) Random authenticated triples, $([a]_S, [b]_S, [c]_S)$, $c = a \cdot b$, used to multiply two shared values.

During the online protocol the circuit is evaluated gate by gate on shared values and using the linearity of the $[\cdot]_S$-representation. To share an input x_i, party P_i takes a pre-processed random value $[r]_S$ and broadcast the value $x_i - r$. Since r is uniformly random in $\mathbb{F}$ and unknown to all other parties, it acts as a one-time pad to perfectly hide x_i. All parties can then locally compute $[r]_S + (x_i - r)$ to obtain $[x_i]_S$.

Multiplication of two shared values $[x]_S$ and $[y]_S$ uses Beaver's trick. Using a multiplication triple $[a]_S, [b]_S, [c]_S$, first parties open and recover the values $\epsilon = x - a$ and $\rho = y - b$. Again, the triple values perfectly mask the inputs x and y, and the opened values appear uniformly random to corrupt parties. Given ϵ and ρ, a sharing of the product $x \cdot y$ can be locally computed by all parties using the triple as follows:

$$[x \cdot y]_S = [c]_S + \epsilon \cdot [b]_S + \rho \cdot [a]_S + \epsilon \cdot \rho.$$

When the circuit evaluation is completed, parties check the MACs on all the values revealed during the input and non-linear operations. If the check passes, they open and recover the output, otherwise the protocol aborts.

Online protocol

Initialize . Parties call $\mathcal{F}_{\mathsf{Prep}}$ to get the shares $\Delta^{(i)}$ of the MAC key, multiplication triples $([a]_S, [b]_S, [c]_S)$ and mask values $(r_i, [r_i]_S)$ as needed for the function under evaluation. If $\mathcal{F}_{\mathsf{Prep}}$ aborts then the parties output $\perp$ and abort.

Input. To share an input x_i, party P_i takes an available mask value $(r_i, [r_i]_S)$ and does the following:

1. Broadcast $\epsilon \leftarrow x_i - r_i$.
2. The parties compute $[x_i]_S$ as $[r_i] + \epsilon$.

Add. On input $([x]_S, [y]_S)$, locally compute $[x + y]_S \leftarrow [x]_S + [y]_S$.

Multiply. On input $([x]_S, [y]_S)$, the parties do the following:

1. Take one multiplication triple $([a]_S, [b]_S, [c]_S)$, compute $[\epsilon]_S \leftarrow [x]_S - [a]_S$, $[\rho]_S \leftarrow [y]_S - [b]_S$. Open those values and run MAC Check.
2. Use Beaver's trick described above.

Output. To output a share $[y]_S$, do the following:

1. Run MAC Check with input all opened values so far. If it fails, output $\perp$ and abort.
2. Open and MAC Check $[y]_S$. If the check fails, output $\perp$ and abort, otherwise accept y as a valid output.

5 SPDZ Pre-processing

Here we show different ways of implementing the pre-processing phase. We recall, once again, that the main (basic) tasks of this step is to produce the following type of random authenticated values:

Input mask. $([r]_S, P_i)$, with the value r known by P_i

Triples. $([a]_S, [b]_S, [c]_S)$, where $c = a \cdot b$

Of course it is possible to pre-process different types of correlated randomness, such as random bits, squares, etc, that can help to improve the efficiency of certain online operations. However, explaining this kind of optimization is out of the scope of this work.

5.1 Pre-processing using threshold homomorphic encryption

As mentioned before, SPDZ offline protocol is based on a 1-leveled threshold homomorphic encryption scheme (introduced in Section 2.1), supporting $O(n)$ additions and one homomorphic multiplication, instantiated with BGV. Let us assume that $\mathcal{M} = \mathbb{F}^N$, where N is the packing parameter. This allows to produce many correlated random values in parallel. The original SPDZ paper, and several subsequent related works, assume a trusted setup $\mathcal{F}_{\text{KeyGen}}$ for the KeyGen algorithm. We recall that this algorithm securely provides to the parties the BGV public key pk and a sharing $\langle \mathsf{sk} \rangle$ of the secret key sk. [DKL+13] describes a covertly secure protocol that achieves this task, and only recently Rotaru et al. [RST+19] have introduced a protocol that implements $\mathcal{F}_{\text{KeyGen}}$ with active security. This protocol is based on oblivious transfer and, specifically, on the MASCOT protocol [KOS16]. The interested reader can find the implementation of the so-called "SPDZ setup functionality" in [RST+19], here we make use of ideal functionality $\mathcal{F}_{\text{KeyGen}}$ in the description of the pre-processing protocol.

Other than $\mathcal{F}_{\text{KeyGen}}$, we also assume another ideal functionality, $\mathcal{F}_{\text{DistDec}}$, that extends the standard BGV decryption algorithm to securely allow distributed decryption inside SPDZ. Now we give an overview of the pre-processing protocol, and later we provide and discuss it in greater detail.

High level description. The passive version of the pre-processing protocol works as follows. Let us assume that the parties have $(\mathsf{pk}, \langle \mathsf{sk} \rangle)$ and $(\langle \Delta \rangle, \mathfrak{ct}_\Delta)$, where $\mathfrak{ct}_\Delta$ is an encryption of the MAC key Δ using BGV.

- To create an input mask $([r]_S, P_i)$, each party P_i samples a random value r and creates a random sharing $\langle r \rangle$, that is P_i sends the relative share $r^{(j)}$ to P_j, for each $j \neq i$. Parties then locally compute the ciphertexts $\mathfrak{ct}_{rj}$ using the common public key pk and broadcast them. Using the homomorphic properties of BGV, parties can locally compute $\mathfrak{ct}_r$ and $\mathfrak{ct}_r \cdot \mathfrak{ct}_\Delta = \mathfrak{ct}_{\mathsf{m}_r}$, i.e. encryptions of the mask r and its MAC. Using a distributed decryption algorithm with $\langle \mathsf{sk} \rangle$, each party obtains a share of m_r. Note that this step requires interaction. The output of this simple procedure is used in the

Input step of the online evaluation to mask the actual input value, as described in Section 4.

- A similar technique is used to produce triples. Each party samples random shares $a^{(i)}, b^{(i)}$ and broadcast the corresponding ciphertexts $\mathfrak{ct}_{a^i}, \mathfrak{ct}_{b^i}$. Parties can compute $\mathfrak{ct}_c, \mathfrak{ct}_{\mathsf{m}_a}, \mathfrak{ct}_{\mathsf{m}_b}$ as before and using the distributed decryption, the MAC sharing $\langle \mathsf{m}_a \rangle$ and $\langle \mathsf{m}_b \rangle$. Since we allow only one homomorphic multiplication, to produce $\langle \mathsf{m}_c \rangle$ parties first decrypt $\mathfrak{ct}_c$, and, with $\langle c \rangle$, they produce a fresh encryption $\tilde{\mathfrak{ct}}_c$ of c that can then be multiplied by $\mathfrak{ct}_\Delta$.

Unfortunately, this simple protocol is not sufficient against active corruptions. Indeed, corrupt parties have the freedom to generate incorrect ciphertexts containing maliciously chosen noise or unknown plaintexts, that would result either in selective failure attacks or information leakage during distributed decryption. To solve this problem SPDZ uses zero-knowledge proofs of plaintext knowledge for every sent ciphertext, to prove that it is correctly generated. A second issue arises in the distributed decryption itself. During this interactive procedure an adversary might add errors both to triples and MAC values. While correctness of triples is checked through an additional check, called "sacrifice" (that we will describe later), errors on MACs have no impact on protocol security as potential errors cause the MAC Check to fail except with negligible probability.

5.1.1 SPDZ zero-knowledge proofs

As mentioned before, to achieve active security SPDZ uses zero-knowledge proof of plaintext knowledge in order to prove that the ciphertexts used to generate pre-processed randomness are correctly generated. While it would be very convenient, in terms of efficiency, to avoid these expensive proofs all together, they seem to be quite unavoidable if we do not want to occur in decryption failures and information leakage both in the pre-processing and, more importantly, in the online computation. Zero-knowledge proofs constitute the main bottleneck in SPDZ implementations, both in terms of communication and runtime. For this reason a consistent amount of work have been devoted to the optimization of those proofs [BBC+18, BCS19, BDLN16, BDTZ16, dL17, KPR18]. Here we informally describe the main idea of these proofs and explain why they are so expensive. For the details the reader may refer to [BCS19, BDLN16].

Roughly, in SPDZ ZK proofs, each party P_i, acting as a prover P, has to prove knowledge of a short preimage x of a linearly homomorphic function f such that $f(x) = y$ and $\|x\| \leq B$, for some bound B. Here f is the BGV encryption function and y is the ciphertext that P_i has to prove being correctly generated. More formally, we need a zero-knowledge proof of knowledge for the relation

$$\mathcal{R}_{ZK} = \{(x, y) \mid y = f(x) \wedge \|x\| \leq B\}.$$

This kind of proofs usually consist of a standard Σ-protocol:

1. P samples a random r such that $\|r\| \leq \tau \cdot B$, for τ sufficiently large (see below), and sends $f(r) = a$ to the verifier V;

2. V samples a random challenge $e \in \{0, 1\}$ and sends it to P;

3. P replies with $z = r + e \cdot x$.

Finally, the verifier checks whether $f(z) = a + e \cdot y$ and that $\|z\| \leq \tau \cdot B$.

It is evident that the bound proven above is not tight. Indeed a sufficiently large τ is necessary to make the distribution of z statistically independent of x and hence provide (honest-verifier) zero-knowledge. Also, we can extract the witness x (and get special soundness) from two correct transcripts $(a, e, z), (a, e + 1, z')$ that a cheating prover can provide, by $f(z - z') = y$, so that $\|z - z'\| \leq 2 \cdot \tau \cdot B$. The term $2 \cdot \tau \cdot B$ is known as *soundness slack* and quantifies the difference between the bound used by an honest prover and what we can force a cheating prover to do.

In short, this approach has two main drawbacks. Firstly, it needs to be repeated many times to reach a sufficiently small soundness. Secondly, a large soundness slack implies in SPDZ larger parameters in the underlying BGV cryptosystem, with consequences in terms of computation and also communication as these ciphertexts need to be sent to all parties in the protocol. As described in [BCS19], the slack can be removed by a modulus switch operation after the ZK proof is executed. Loosely speaking, a modulus switching operation is a noise management technique, introduced by Brakerski et al. [BGV12], that transforms a ciphertext over a certain modulo into a ciphertext defined over a smaller modulo.

A common solution to the first issue is to use standard amortized techniques [CD09], and prove several statements at once. Even if on one hand the amortization reduces the soundness from $1/2$ to 2^{-t}, where t is the number of instances we are proving, on the other hand it introduces even more slack.

Different alternatives to this general approach have been proposed.

In [DKL+13] a cut-and-choose based check is described to replace the zero-knowledge proofs. This method needs a large number of additional ciphertexts and it seems to require too much memory to be practical.

In [KPR18], Keller, Pastro and Rotaru revisited the original SPDZ ZK proofs by noting that in the pre-processing it is not required that each ciphertext $\mathfrak{ct}_{x_i}$ was correctly generated, but rather than the sum of those is "correct". This is because only this sum is going to be used in the distributed decryption, and not the single shares. In this way it is possible to replace the per-party proof with a global proof, improving the overall computational complexity, as each party needs to check only a single proof instead of $n - 1$, but not the overall communication.

In a recent work [BCS19], Baum, Cozzo and Smart improve the soundness of the global proof introduced in [KPR18]. This work implies a reduction in the amount of amortization required to achieve the desired soundness and also smaller slack. With this technique it is only possible to prove the validity of ciphertexts $2 \cdot \mathfrak{ct}_x$, and not of $\mathfrak{ct}_x$, but in SPDZ this can be mitigated by slightly modifying some of the shares in the MPC protocol.

5.1.2 SPDZ sacrifice

To ensure triples correctness essentially all SPDZ-style protocols use a standard sacrifice technique that checks a pair of triples such that one can be then used securely. While the original SPDZ protocol used two independent random triples $([a]_S, [b]_S, [c]_S)$ and $([a']_S, [b']_S, [c']_S)$, checking one against the other, in MASCOT it was noticed that the check also works with "correlated" triples $([a]_S, [b]_S, [c]_S)$ and $([a']_S, [b]_S, [c']_S)$, i.e. with the same b (or equivalently same a). In this way we have a cheaper check requiring less authenticated randomness and also less opening, and hence less communication. It proceeds as follows.

Sacrifice: Given two correlated triples $([a]_S, [b]_S, [c]_S)$ and $([a']_S, [b]_S, [c']_S)$:

1. Parties call the ideal functionality $\mathcal{F}_{\mathsf{Rand}}$ to obtain a random $r \in \mathbb{F}$
2. Parties open the value $\rho = r \cdot [a] - [a']$
3. Parties compute $r \cdot [c] - [c'] - \rho \cdot [b]$, and check whether it is equal to zero. If not, the protocol outputs $\perp$ and aborts.

If the triples are correct:

$$r \cdot c - c' - \rho \cdot b = r \cdot (a \cdot b) - (a' \cdot b) - b \cdot (r \cdot a - a') = 0.$$

If the triples are incorrect, that is $(a, b, c + \delta)$ and $(a', b, c' + \delta')$, where δ, δ' are chosen by the adversary:

$$\begin{aligned} &r \cdot (c + \delta) - (c' + \delta') - \rho \cdot b = \\ &r \cdot (a \cdot b + \delta) - (a' \cdot b + \delta') - b \cdot (r \cdot a - a') = r \cdot \delta - \delta', \end{aligned}$$

which is zero with probability $1/|\mathbb{F}|$.

5.1.3 Putting everything together

We can finally show the SPDZ offline protocol. We make the following assumptions:

- A global zero-knowledge protocol Π_{gZKPoK} as we have described above (for details we refer to [BCS19, KPR18, OSV20].
- A key generation functionality $\mathcal{F}_{\text{KeyGen}}$ that distributes $(\mathsf{pk}, \langle \mathsf{sk} \rangle)$ among the parties (in SPDZ instantiation these keys are BGV encryption and decryption keys).
- A distributed decryption functionality $\mathcal{F}_{\text{DistDec}}$ that, given a correctly generated ciphertext, output a sharing of the decryption output.

When we implement $\mathcal{F}_{\text{DistDec}}$ in SPDZ using BGV, we provide this ideal functionality of two commands **DDM** and **DDT**, that we describe below.

Distributed decryption MACs (DDM): It takes as input a valid ciphertext $\mathfrak{ct}_m$ and the BGV keys $(\mathsf{pk}, \mathsf{sk})$.

1. Decrypt $\mathfrak{ct}_m$ and send the output of the decryption m to the adversary.
2. Wait for an input from the adversary. If receive Abort, send Abort to the parties and halt, otherwise on receiving $m' = m + \delta$, send $\langle m' \rangle$ to the parties.

Distributed decryption triples (DDT): It takes as input a valid ciphertext $\mathfrak{ct}_m$ and the BGV keys $(\mathsf{pk}, \mathsf{sk})$.

1. Do as **DDM** in steps 1. and 2.
2. Compute a fresh encryption $\mathfrak{ct}_{m'}$ of m' and send it to the parties.

DDT is essentially the $\mathsf{Reshare}$ protocol given in [DKL+13, DPSZ12, **?**]. When we implement this functionality with BGV, the protocol requires a masking ciphertext, and hence a ZK proof, that is used in the distributed decryption. Other than the decryption sharing $\langle m' \rangle$, it also produces a fresh encryption $\mathfrak{ct}_{m'}$ of the output of the decryption. On the other hand, in the protocol implementing DDM [KPR18], a large "plaintext" mask is introduced directly in the decryption procedure, and there is no need of ZK proof for this mask. Therefore, this latter protocol is cheaper than DDT, but it can only be used if the result of the decryption does not need to be re-encrypted. In particular, it can only be used for MACs generations and not for generating c and m_c, and for this reason it only outputs decryption shares and not a fresh encryption of it as DDT does. Finally, note that both of the commands allow the corrupt parties to add some error to the outputs. This does not break the security of the protocol as these errors will be detected by MAC Check failures.

Pre-processing protocol. Parties receive pk from $\mathcal{F}_{\text{KeyGen}}$.

Initialize. Parties create a ciphertext $\mathfrak{ct}_\Delta$ encrypting the MAC key Δ:

1. Each party P_i samples a random $\Delta^{(i)}$. Set $\Delta = \sum_i \Delta^{(i)}$
2. Each $P_i, i \in [n]$, computes and broadcasts $\mathfrak{ct}_{\Delta^i}$
3. Parties run Π_{gZKPoK} to check that $\mathfrak{ct}_\Delta$ is valid

Input. On input (Input, P_i) from all parties:

1. P_i samples $r \leftarrow \mathbb{F}$, creates $\langle r \rangle$ and sends $r^{(j)}$ to $P_j, j \neq i$
2. Each party P_i creates $\mathfrak{ct}_{r^i}$ and broadcasts this value
3. Parties run Π_{gZKPoK} and compute $\mathfrak{ct}_{r \cdot \Delta}$
4. Parties call $\mathcal{F}_{\text{DistDec}}$ on command **DDM** receiving $\langle \mathsf{m}_r \rangle$
5. Parties run MAC Check, if it fails, the protocol Abort

Triples. On input (Triple) from all parties:

1. Each P_i samples random shares $a^{(i)}, b^{(i)} \leftarrow \mathbb{F}$, computes $\mathfrak{ct}_{a^i}, \mathfrak{ct}_{b^i}$ and broadcasts these values
2. Parties run Π_{gZKPoK} to check the validity of $\mathfrak{ct}_a$ and $\mathfrak{ct}_b$

3. Parties compute $\mathfrak{ct}_a \cdot \mathfrak{ct}_b = \mathfrak{ct}_c$
4. Parties call $\mathcal{F}_{\text{DISTDEC}}$ on command **DDT** obtaining $\langle c \rangle$ and a fresh encryption of c, $\tilde{\mathfrak{ct}}_c$
5. Parties obtain $\langle \mathsf{m}_a \rangle, \langle \mathsf{m}_b \rangle, \langle \mathsf{m}_c \rangle$ calling $\mathcal{F}_{\text{DISTDEC}}$ on command **DDM** on inputs $\mathfrak{ct}_a, \mathfrak{ct}_b, \tilde{\mathfrak{ct}}_c$.
6. Parties repeat steps 2-5 with value a', obtaining $\langle c' \rangle$, such that $c' = a' \cdot b$ and $\langle \mathsf{m}_{a'} \rangle, \langle \mathsf{m}_{c'} \rangle$.
7. Parties run the **Sacrifice** check on input $\big((a, b, c), (a', b, c')\big)$. If the check fails, the protocol Abort
8. Parties run MAC Check, if it fails, the protocol Abort

5.2 Pre-processing using oblivious transfer

Here we describe how to generate random authenticated values and triples using oblivious transfer instead of homomorphic encryption. In order to do this we need some more notation.

We define the 'gadget' vector $\mathbf{g}$ consisting of the powers of two (in $\mathbb{F}_p$) or powers of X (in extension fields $\mathbb{F}_{p^k}$), so that

$$\mathbf{g} = (1, g, g^2, \ldots, g^{k-1}) \in \mathbb{F}^k,$$

where, as said before, $g = 2$ in $\mathbb{F}_p$ and $g = X$ in $\mathbb{F}_{p^k}$. Let $\mathbf{g}^{-1} : \mathbb{F} \to \{0,1\}^k$ be the 'bit decomposition' function that maps $x \in \mathbb{F}$ to a bit vector $\mathbf{x}_B = \mathbf{g}^{-1}(x) \in \{0,1\}^k$, such that $\mathbf{x}_B$ can be mapped back to $\mathbb{F}$ by taking the inner product $\langle \mathbf{g}, \mathbf{g}^{-1}(x) \rangle = x$. This tool permits to switch between field elements and vectors of bits whilst remaining independent of the underlying finite field.

Passively-secure multiplication using OT. We are now ready to show how to use OT to produce a secret sharing of an arithmetic product. In a standard one-out-of-two OT, the sender inputs two messages $x_0, x_1 \in \mathbb{F}$, and the receiver inputs a bit b, receiving $x_b = x_0 + b \cdot (x_1 - x_0)$. Setting $a = x_1 - x_0$, we obtain

$$x_b - x_0 = b \cdot a,$$

where $x_b, x_0, a \in \mathbb{F}$ and $b \in \{0,1\}$. The value a is called *correlation*, and the corresponding OT functionality, *correlated OT* (Figure 3).

We can then combine k correlated OTs into one arithmetic OT, as follows. Parties P_S and P_R input $(x_i, x_i + a)$, for some fixed correlation $a \in \mathbb{F}$, and $(b_1, \ldots, b_k)$, such that $(b_1, \ldots, b_k) = \mathbf{g}^{-1}(b)$, $b \in \mathbb{F}$, respectively. The receiver then obtains $y_i = x_i + b_i \cdot a$, $i \in [k]$. By setting $q = \langle \mathbf{g}, \mathbf{y} \rangle$, with $\mathbf{y} = (y_1, \ldots, y_k)$ and $t = \langle \mathbf{g}, \mathbf{x} \rangle$, with $\mathbf{x} = (x_1, \ldots, x_k)$, we obtain $q = t + b \cdot a$, where the sender holds $q, a \in \mathbb{F}$ and the receiver holds $t, b \in \mathbb{F}$. We have thus transformed oblivious transfer into a secret sharing of the product of both parties' inputs

Functionality $\mathcal{F}_{\mathsf{COT}}$

Running between a sender P_S and a receiver P_R, it operates as follows.

- P_S inputs $(x_0, x_0 + a) \in \mathbb{F} \times \mathbb{F}$ and P_R inputs b.
- The functionality outputs $x_b = x_0 + b \cdot a$ to P_R.

Figure 3: Functionality for one-out-of-two oblivious transfers on k-bit strings.

in $\mathbb{F}$. [3] Using this building block, constructing a passively secure protocol for secret-shared multiplication triples is straightforward by simply running the protocol between every pair of parties and summing the shares.

Efficient authentication using correlated OT. As for the case of homomorphic encryption, also in oblivious transfer based pre-processing protocols we can use the same approach to create triples and MACs, because the relation between authenticated values and MAC keys is the same as the multiplication triple relation. The main difference is that in an authentication procedure, the global MAC key is fixed, so while in triples generation we need to use a fresh correlation for each triple, the correlation remains the same for all the values we need to authenticate. More precisely, the MAC generation for an additively secret shared value $x \in \mathbb{F}$ proceeds as follows.

1. Each party P_i samples a random share $\Delta^{(i)}$ of the global MAC key
2. Each pair of parties, (P_i, P_j), run k $\mathcal{F}_{\mathsf{COT}}$ on input $x^{(i)}$, $\Delta^{(j)}$, respectively, obtaining
$$q^{(j,i)} = t^{(i,j)} + x^{(i)} \cdot \Delta^{(j)}.$$
3. After all the $n(n-1)$ executions, each party P_i locally combines their results to generate the MAC share
$$m^{(i)} = x^{(i)} \cdot \Delta^{(i)} + \sum_{j \neq i} \left(q^{(i,j)} - t^{(i,j)} \right).$$

Essentially, using $\mathcal{F}_{\mathsf{COT}}$, i.e. a 2-party functionality, we naturally obtain a BDOZ-style authentication that can be locally converted to SPDZ-style MACs as we described in Section 3.

OT-based pre-processing with active security. It is clear, from previous description of the authentication and triple generation protocols, that an adversary could easily cheat, for example by inputting inconsistent values in one of the $\mathcal{F}_{\mathsf{COT}}$ instances, or using different MAC key shares with different parties.

[3] This generalisation of oblivious transfer is also referred to as *oblivious linear function evaluation* (OLE) [NP99].

Here we discuss separately how to achieve active security of the MACs generation and triples generation protocols.

For the MAC generation, it turns out that the passively secure protocol is almost enough. This is because during the authentication, the correlation Δ is fixed at the beginning, so the adversary does not have much possibility to deviate from protocol instructions later on. However, even after the correlation has been fixed, the adversary is still able to create wrong MACs which contain errors depending on the global key. It was proved in [KOS16], that to obtain active security it is enough to run a MAC Check opening a random linear combination of authentication values just after their generation. This somehow fixes the global key and ensures correctness of subsequent checks. Note that during the MAC checks an adversary is still able to pass the check even in the presence of some errors by guessing some bits of Δ, however if the guess is incorrect the protocol aborts. So the only thing we have to make sure is that the global MAC key still has sufficient entropy to prevent cheating in MAC checks, even if a few bits have been guessed.

For triple generation achieving active security is more involved, since we do not have a fixed correlation, and hence a linear combination on which running a check. Note that we need to ensure both correctness and privacy of the triples. Correctness is easily verified with a pairwise, standard sacrifice technique. This check, however, raises the possibility of selective failure attacks, so that if for example the adversary cheats in just a single bit, and the check passes, then this bit of the triple is leaked to the adversary. To prevent this, a simple variant of privacy amplification is used. First we generate several leaky triples, from which a single, random triple is extracted by taking random combinations [KOS16].

5.2.1 SHE vs OT - Comparison

Here we compare the efficiency of OT-based and HE-based pre-processing, reporting the figures provided by [KPR18]. The values in Table 1 confirm the complementarity of these two approaches, even if some recent improvements in OT-extension protocols could greatly improve the efficiency of protocols relying on oblivious transfer. We can see that MASCOT is more efficient over binary extension fields and LowGear over prime fields of odd characteristics. In the multiparty case, essentially when the number of parties is larger than ~ 7, HighGear will become more efficient than LowGear [KPR18].

5.2.2 Pre-processing with OLE

We can naturally instantiate the pre-processing with OLE (Oblivious Linear-function Evaluation) instead of OT. As we said previously, OLE is an arithmetic generalization of OT to larger fields. More formally, it is a two-party functionality where the sender P_S inputs two values $a, b \in \mathbb{F}$ and the receiver P_R inputs a value $x \in \mathbb{F}$ obtaining $y = x + a \cdot b$. OLE can be constructed from several assumptions and public-key based constructions, like OT (as seen before), homomorphic encryption, noisy encodings [IPS09, GNN17], etc. An efficient arithmetic implementation of OLE can potentially lead to a very efficient pre-processing phase, as it will avoid running $\mathcal{F}_{\mathsf{COT}}$ for each bit of the binary representation of the values

Protocol	Triples/sec	Network	Field
MASCOT	5100	1Gbit/s	Prime field $\log_2 \lvert\mathbb{F}\rvert = 128$
	214	50Mbit/s	Prime field $\log_2 \lvert\mathbb{F}\rvert = 128$
	5100	1Gbit/s	Binary field $\mathbb{F}_{2^{128}}$
LowGear	**30000**	1Gbit/s	Prime field $\log_2 \lvert\mathbb{F}\rvert = 128$
	3200	50Mbit/s	Prime field $\log_2 \lvert\mathbb{F}\rvert = 128$
	117	1Gbit/s	Binary field $\mathbb{F}_{2^{128}}$
HighGear	5600	1Gbit/s	Prime field $\log_2 \lvert\mathbb{F}\rvert = 128$
	1300	50Mbit/s	Prime field $\log_2 \lvert\mathbb{F}\rvert = 128$
	67	1Gbit/s	Binary field $\mathbb{F}_{2^{128}}$

Table 1: Triple generation for prime and binary fields with two-party and 64 bits of statistical security [KPR18].

involved in the computation. A two-party protocol based on OLE is described by Döttling et al. [DGN+17], that can be considered as a natural generalization of TinyOT to the arithmetic setting, however this work does not give an implementation of the protocols described, so the actual efficiency of this approach is not completely clear.

5.3 Silent pre-processing via PCG

In a recent line of work Boyle et al. [BCG+19b, BCG+20] show how to generate correlated randomness that can be used as pre-processd material in MPC protocols using *pseudorandom correlation generators* (PCGs). A PCG is a deterministic function that allows to extend short seeds to long instances of a desired correlation, i.e. OT, OLE, triples etc. Using a PCG we can have a so-called "silent pre-processing". After a setup that consists of generation and distribution of the seeds, the expansion is local, i.e. does not require communication, and hence the term "silent".

This is a very promising approach as it allows to reduce significatively the communication and memory usage, even if it still require, in some useful case like generation of authenticated triples, a quite expensive setup.

References

[ACC+] Abdel Aly, Kelong Cong, Daniele Cozzo, Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Oliver Scherer, Peter Scholl, Nigel P. Smart, Titouan Tanguy, and Tim Wood. Scale - mamba v1.9: Documentation.

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead.

In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 223–254. Springer, Heidelberg, August 2017.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, October 2011.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012.

[ALSZ13] 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*, pages 535–548. ACM Press, November 2013.

[ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 673–701. Springer, Heidelberg, April 2015.

[AOR+19] Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P. Smart, and Tim Wood. Zaphod: Efficiently combining LSSS and garbled circuits in SCALE. In Michael Brenner, Tancrède Lepoint, and Kurt Rohloff, editors, *Proceedings of the 7th ACM Workshop on Encrypted Computing and Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, November 11-15, 2019*, pages 33–44. ACM, 2019.

[BBC+18] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 669–699. Springer, Heidelberg, August 2018.

[BCG+19a] 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*, pages 291–308. ACM Press, November 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019,*

Part III, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Heidelberg, August 2020.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BCS19] 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*, volume 11959 of *LNCS*, pages 274–302. Springer, Heidelberg, August 2019.

[BDLN16] Carsten Baum, Ivan Damgård, Kasper Green Larsen, and Michael Nielsen. How to prove knowledge of small secrets. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 478–498. Springer, Heidelberg, August 2016.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

[BDTZ16] Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Winther Zakarias. Better preprocessing for secure multiparty computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 327–345. Springer, Heidelberg, June 2016.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, Heidelberg, August 1995.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

[Ben18] Aner Ben-Efraim. On multiparty garbling of arithmetic circuits. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2018.

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016.

[BOS16] Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 461–490. Springer, Heidelberg, October / November 2016.

[BOSS20] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 562–592. Springer, Heidelberg, August 2020.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Heidelberg, August 2012.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 395–426. Springer, Heidelberg, August 2018.

[CD09] Ronald Cramer and Ivan Damgård. On the amortized complexity of zero-knowledge protocols. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 177–191. Springer, Heidelberg, August 2009.

[CDE+18] 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, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 769–798. Springer, Heidelberg, August 2018.

[CDFG20] 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, Part II*, volume 12111 of *LNCS*, pages 357–386. Springer, Heidelberg, May 2020.

[CDv88] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 87–119. Springer, Heidelberg, August 1988.

[CFY17] Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. Catching MPC cheaters: Identification and openability. In Junji Shikata, editor, *ICITS 17*, volume 10681 of *LNCS*, pages 110–134. Springer, Heidelberg, November / December 2017.

[CG20] Ignacio Cascudo and Jaron Skovsted Gundersen. A secret-sharing based mpc protocol for boolean circuits with good amortized complexity. 2020. https://eprint.iacr.org/2020/162.

[CKR+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. *IACR Cryptol. ePrint Arch.*, 2020:451, 2020.

[CL14] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 466–485. Springer, Heidelberg, December 2014.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[Cou19] Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 473–503. Springer, Heidelberg, May 2019.

[DEF+19] 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. IEEE Computer Society Press, May 2019.

[DGN+17] Nico Döttling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto Trifiletti. TinyOLE: Efficient actively secure two-party computation from oblivious linear function evaluation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2263–2276. ACM Press, October / November 2017.

[DKL+13] 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*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

[DKS+17] 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 *NDSS 2017*. The Internet Society, February / March 2017.

[dL17] Rafaël del Pino and Vadim Lyubashevsky. Amortization with fewer equations for proving knowledge of small secrets. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 365–394. Springer, Heidelberg, August 2017.

[DLT14] Ivan Damgård, Rasmus Lauritsen, and Tomas Toft. An empirical study and some improvements of the MiniMac protocol for secure computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 398–415. Springer, Heidelberg, September 2014.

[DNNR17] 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, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 167–187. Springer, Heidelberg, August 2017.

[DO10] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In Tal Rabin, editor,

CRYPTO 2010, volume 6223 of *LNCS*, pages 558–576. Springer, Heidelberg, August 2010.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015*. The Internet Society, February 2015.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of Boolean circuits using preprocessing. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 621–641. Springer, Heidelberg, March 2013.

[EGK+20] 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, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 823–852. Springer, Heidelberg, August 2020.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[FKOS15] 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, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 711–735. Springer, Heidelberg, November / December 2015.

[FPY18] Tore K. Frederiksen, Benny Pinkas, and Avishay Yanai. Committed MPC - maliciously secure multiparty computation from homomorphic commitments. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 587–619. Springer, Heidelberg, March 2018.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/2012/144.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *46th ACM STOC*, pages 495–504. ACM Press, May / June 2014.

[GIW16] Daniel Genkin, Yuval Ishai, and Mor Weiss. Binary AMD circuits from secure multiparty computation. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 336–366. Springer, Heidelberg, October / November 2016.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, July 2005.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GNN17] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 629–659. Springer, Heidelberg, December 2017.

[HHNZ19] 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*, pages 1220–1237. IEEE Computer Society Press, May 2019.

[HKS+10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 451–462. ACM Press, October 2010.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 598–628. Springer, Heidelberg, December 2017.

[HVW20] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. The price of active security in cryptographic protocols. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 184–215. Springer, Heidelberg, May 2020.

[IKM+13] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 600–620. Springer, Heidelberg, March 2013.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors,

CRYPTO 2014, Part II, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, August 2014.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 294–314. Springer, Heidelberg, March 2009.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st ACM STOC*, pages 44–61. ACM Press, May 1989.

[JL13] Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 76–92. Springer, Heidelberg, May 2013.

[Kel20] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. *IACR Cryptol. ePrint Arch.*, 2020:521, 2020.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013.

[KOR+17] 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*, volume 10355 of *LNCS*, pages 229–249. Springer, Heidelberg, July 2017.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015.

[KOS16] 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*, pages 830–842. ACM Press, October 2016.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

[KSS13] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. *J. Comput. Secur.*, 21(2):283–315, 2013.

[LOS14] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 495–512. Springer, Heidelberg, August 2014.

[LPO11] Yehuda Lindell, Benny Pinkas, and Eli Oxman. The IPS compiler: Optimizations, variants and concrete efficiency. Cryptology ePrint Archive, Report 2011/435, 2011. http://eprint.iacr.org/2011/435.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, Heidelberg, August 2015.

[LSS16] Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 554–581. Springer, Heidelberg, October / November 2016.

[MR18] 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*, pages 35–52. ACM Press, October 2018.

[MW19] Eleftheria Makri and Tim Wood. Full-threshold actively-secure multiparty arithmetic circuit garbling. Cryptology ePrint Archive, Report 2019/1098, 2019. https://eprint.iacr.org/2019/1098.

[NNOB12] 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, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *31st ACM STOC*, pages 245–254. ACM Press, May 1999.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-N OT extension with application to private set intersection. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 381–396. Springer, Heidelberg, February 2017.

[Orl11] Claudio Orlandi. Is multiparty computation any good in practice? In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, May 22-27, 2011, Prague Congress Center, Prague, Czech Republic*, pages 5848–5851. IEEE, 2011.

[OSV20] 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, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 254–283. Springer, Heidelberg, February 2020.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009.

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer. 1981.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[RST+19] Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for SPDZ. Cryptology ePrint Archive, Report 2019/1300, 2019. https://eprint.iacr.org/2019/1300.

[RW19] Dragos Rotaru and Tim Wood. MArBled circuits: Mixing arithmetic and Boolean circuits with active security. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 227–249. Springer, Heidelberg, December 2019.

[RWT+18] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *ASIACCS 18*, pages 707–721. ACM Press, April 2018.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In

Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

[SV14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.*, 71(1):57–81, 2014.

[Wie83] Stephen Wiesner. Conjugate coding. 15(1):78–88, 1983.

[WRK17a] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, October / November 2017.

[WRK17b] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 39–56. ACM Press, October / November 2017.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated ot with small communication. *IACR Cryptol. ePrint Arch.*, 2020:924, 2020.