Milestone Report

P4: Secure and Privacy-Preserving Semiconductor Testing

Sebastian Hasler

October 4th, 2022

1 Introduction

Over the past two decades the semiconductor industry has been shifting from an integrated device manufacturers (IDMs) dominated market towards the fabless model due to the ever-increasing complexity of the semiconductor design processes [HCW17, Als21]. Continuous outsourcing resulted in today's globally distributed semiconductor supply chain with its multitude of highly specialized service providers such as foundries, fabless designers, test houses, and Electronic Design Automation (EDA) software providers.

During the semiconductor production process, large amounts of data are generated and have to be analyzed and monitored, e.g., data from fabrication equipment, measurements from the testing process or supplementary data from managing the production process itself. In addition to its actual role in the respective part of the production process, this data is also used in machine learning (ML) algorithms to optimize the production process. The success and accurancy of ML thereby heavily depends on the quality and the amount of available data.

However, a large amount of the currently generated data is sensitive and the data owners are unwilling to share this data without security and privacy guarantees, since it contains insights into business-critical information such as yield, characteristics of devices, and operational parameters. This information needs to be protected from unwanted disclosure or industrial espionage. Besides intellectual property (IP) issues, the data may also involve customer data which cannot be forwarded to third parties legally.

To date, in lack of better solutions, confidentiality of data is usually approached by contractual means, such as NDAs or consenting on trusted third parties. This practice can less and less meet modern security and privacy standards and growing confidentiality concerns [IEE20].

Privacy-Preserving Computations. The sketched problem calls for privacy-preserving alternatives. The goal of this project is to enable the parties in the semiconductor production chain to carry out computations on each other's data while each party's input remains private. We emphasize that this goal cannot be achieved by access control mechanisms alone. With access control parties either get access to private data or not. We however want parties to be able to compute on private data without ever getting access to the private input data itself.

There exist several privacy-enhancing and privacy-preserving technologies one could use and try to integrate into the semiconductor production chain in order to achieve our goal, most notably: Multi-Party Computation (MPC) [DPSZ12], Differential Privacy (DP) [Dwo06], Fully Homomorphic Encryption (FHE) [Gen09], and Trusted Execution Environments (TEEs) [Sch16].

In our setup, MPC is a natural choice given the many different parties involved in the semiconductor production chain. MPC allows two or more parties to compute a function on private inputs without revealing the inputs or intermediate results to the other parties or observers.

MPC guarantees that every party learns only the output intended for this party. It guarantees the privacy of each party's data even against malicious competitors and is efficient enough to be applied in large-scale industrial solutions [ABL+18]. Furthermore, MPC has been successfully used for privacy-preserving ML [RRK18, ZPGS19, KRC+20, CPR19, CKR+20] and therefore is a promising candidate in the semiconductor production chain.

We base our approach on state-of-the-art secret-sharing-based MPC protocols [DPSZ12, KOS16, KPR18] and their implementations, e.g. [Kel20], and adapt them to the special requirements of the production chain. These kinds of protocols (and therewith our adaption) come with an efficient two-phase approach which is particularly useful for regularly timed computations like they occur in standardized production processes. An input-independent offline phase generates standardized structured random data between the actual computations. The input-dependent online phase then uses this preprocessed random data to carry out the actual computation on sensitive data very efficiently. Due to their versatility, efficiency, and high security standards we currently consider this MPC approach to be best suited for applications in the semiconductor industry.

Once the underlying MPC protocol is implemented it can be employed directly by the industry partners to compute on private data. We analyze this *direct approach* in Section 2.2 and compare it to a more evolved *cloud-based approach*. In the cloud-based setup the actual MPC is run by two or more independent servers, the industrial players only provide the inputs to the servers (without the servers and other players learning the actual inputs) and receive the results. Our comparison shows clear advantages of the latter setup for semiconductor applications. However, the cloud-based approach is technically and security-wise more challenging than the direct approach and cannot be naively implemented. We provide a solution to these security challenges based on theoretical work by [DDN+16]. As a result we are able to construct a proof-of-concept automated toolkit that allows the industry partners to integrate the new MPC components into their infrastructure used in the production chain.

Our Contributions so far. As a first step towards privacy-preserving computations in the semiconductor production chain, we make the following contributions:

1. We present a cloud-based MPC solution that can run arbitrary computations for an arbitrary number of (input/output) parties where, as long as at least one cloud server is honest, no party gets any information about the inputs of other parties, except for what they can derive from their outputs.

2. Our cloud-based implementation completely automates the instantiation of MPC protocol runs in the cloud via convenient APIs that can be used by clients. We also implement so-called input and output protocols by Damgård et al. [DDN+16].

3. We evaluate our implementation for semiconductor testing on real-world test data from an ASIC production line by Advantest. The results are analyzed with respect to performance and scalability. We benchmark the online and offline phase of our protocols separately and discuss the advantages of the two-phase MPC approach for applications in the semiconductor production.

Work Program Ahead. Currently, we work on vastly reducing network traffic in the so-called offline phase of MPC, as this emerged as a major way to reduce cost in cloud-based deployments. In our optimized offline phase for MPC over $\mathbb{Z}_{2^k}$, the amortized communication is more than 2x better than the previous best [CKL21] in a setting with two cloud servers. Furthermore, we implemented this offline phase as a Rust software, thus providing the first-ever software implementation of a HE-based offline phase for SPD$\mathbb{Z}_{2^k}$. The improvement is even larger compared to the only other work in the same setting that has been implemented in software, namely the original SPD$\mathbb{Z}_{2^k}$ offline phase implemented in MP-SPDZ [DEF+19]: Our

analysis suggests that Multipars outperforms it by a factor of 8–32x, depending on domain and security parameters. This research is ongoing and will be submitted later this year, after our implementation and benchmarks have been finalized.

While, as mentioned, the main focus in this project so far was cloud-based MPC, an on-premises approach involving ACS Edge was started to be discussed as a platform to run secure computations directly on the test floor. However, the precise infrastructure and exact security and privacy requirements, including trust assumptions, in the semiconductor life cycle still need to be formalized. Based on such a formalization, suitable privacy-preserving methods will be developed and evaluated for ACS Edge based semiconductor testing.

Structure of the Report. In Section 2, we present a concrete scenario for privacy-preserving computations for semiconductor testing. We also discuss the mentioned direct and cloud-based MPC approaches. In Section 3, we then present our concrete implementation of the cloud-based MPC approach, with an evaluation and benchmarks presented in Section 4. Afterwards in Section 5, we sketch our ongoing work that aims to reduce cost and compare it to prior works. We conclude in Section 6.

2 Scenarios for Privacy-Preserving Computations

In this section, we discuss the application potential for privacy-preserving computation in the semiconductor production chain and provide a high-level overview of our approach.

2.1 Example Scenarios

As a common scenario for our use cases, we assume a testing process where a fabless chip designer, an equipment manufacturer, and an outsourced semiconductor assembly and test vendor (OSAT) need to work together to enable a smooth testing process. Nonetheless, each partner has secret information, which she is not willing to share during the computation:

- The main asset of a *fabless company* is the IP and characteristics of the devices. The testing process is based on measuring a wide set of device parameters, which can be used to derive sensitive information. E.g., the yield (i.e., the number of good devices) is sensitive as it allows estimates on the production cost.
- The *test houses or OSATs* operate the testing process and the needed test equipement as a service for the fabless. Consequently, the test results also contain sensitive information about the OSAT such as timestamps or equipment properties. It can allow the fabless to reconstruct internal business parameters of the OSAT, e.g., the overall equipment usage or its efficiency.
- The *test equipment manufacturer* provides and delivers testing machines to the OSAT. Here, equipment internals such as configurations, calibration, and telemetry data need to be protected.

Depending on the actual computation the parties want to perform, only some or all of the above-mentioned sensitive data must be processed. In what follows, we show for three examples from the semiconductor production chain how this can be done in a privacy-preserving manner. Note that we restrict our security analysis to cases where the test equipment and its communication with the chip are uncompromised. Securing the test equipment against compromise is left for future work as well as additionally securing the tester's communication with the chip and therewith prevent unauthorized access to the chip's testing interfaces, cf. [RK10, DSFDNR19].

Test Execution and Monitoring. Test execution can often be reduced to a sequence of comparison operations. For each test, one needs to check if the measurements are within specified limits. In the more general case where the comparison operations can depend on previous comparison results, one needs to evaluate a decision tree on the measurements. The test limits directly represent the specifications of the product. However, running the test process and performing statistical analysis for monitoring traditionally requires processing of the limits by the OSAT. With a privacy-preserving computation the OSAT can run and monitor the testing process without access to the underlying sensitive data.

Equipment Health Monitoring. In this scenario, the fabless and the OSAT have a joint interest in monitoring the employed test equipment. To do so, irregularities in the testing process need to be identified to spot, e.g., equipment that needs maintenance. In the simplest form, the yield is a first level indicator assuming a certain homogeneity in the distribution across all devices of a fabrication lot. Classically, performing such operations requires the fabless and the OSAT to share the measurements of each test cell as well as the utilization details of the test equipment. However, the partners are mutually reluctant to share this information to prevent data leakage and industry espionage. Using a privacy-preserving computation, one can perform computations where the fabless provides the measurements, while the OSAT provides the detailed utilization and telemetry data of each employed equipment, without actually revealing this data to the other party. Using correlation analysis, equipment showing irregularities is identified and returned to the OSAT to perform further investigations.

Privacy-Preserving Machine Learning. In the semiconductor domain big data and ML-based data analytic techniques are increasingly employed [IEE20]. To realize ML algorithms in a privacy-preserving way is itself a highly active research field, ranging from linear and logistic regression [CPR19, ZPGS19] to deep learning [RRK18, CKR+20, KRC+20]. Applications for ML in the semiconductor production include (among others) to predict the later behavior of devices based on early data [YF19, JKK+19] or to improve the classification of devices during tests (i.e., binning) [SKWT17]. However, privacy-preserving ML has not been applied in the semiconductor production chain so far. With our cloud-based MPC system we provide a platform for ML applications, but leave it to future work to specifically target ML in the semiconductor domain.

2.2 Direct Approach

One way to achieve privacy-preserving computation is to run MPC direcly between the parties who own the data. To run the program as an MPC protocol, they first need to agree on the program's source code. We note that the source code can be a program pattern where the actual behavior is largely determined by the input the parties provide.[1] Next, the parties compile the consented program into an arithmetic circuit (consisting of addition and multiplication gates). Finally, they follow an MPC protocol to jointly evaluate the circuit. To communicate the parties use secure unicast channels, such as a TLS connections, between the parties as well as a secure broadcast channel, e.g., some kind of a bulletin board where parties post or fetch messages (see Fig. 1).

2.3 Cloud-Based Approach

Many companies have moved or are in the process of moving their services to the cloud in order to exploit numerous advantages of cloud-computing. These advantages include—but are not

[1] In the most extreme case, the program could be a universal machine which interprets the input it gets as a program that it then runs.

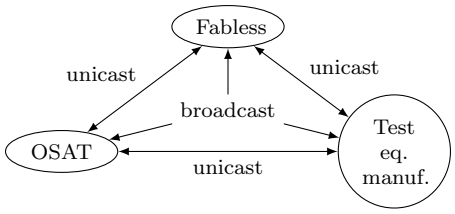

Figure 1: Direct MPC between three parties

limited to—automated management, flexibility, easier scalability, availability, data loss prevention and pay-per-use. Cloud-computing fits particularly well to the new demands that come with big data. By adapting MPC for cloud computing we get additional advantages: differences in computational power of different parties can be avoided, as parties don't need to have the on-premises infrastructure required to run MPC. The cloud-based approach also has much better support for a large and flexible set of parties, and the complexity of its main protocol is independent of the set size.

A naive approach to realize MPC in the cloud setup is the following: each party provides her sensitive data to a cloud server and the cloud servers run the MPC protocol. However, in this scenario, if a cloud provider is malicious, it can misuse, manipulate or leak all of a party's private data. In other words, this naive cloud-based approach considers the cloud providers as trusted parties. We can however adapt MPC in a more clever way to the cloud setup in order to avoid trusted parties such that we can guarantee privacy of parties' data even if only a single provider is honest. We will describe our solution in the following; details on its security guarantees are discussed in Section 3.1.

We start with so-called *clients*, such as fabless companies, OSATs, and test equipment manufacturers (see Fig. 2) who want to compute a certain function together on their respective private inputs. The clients want to receive their output at the end of the joint function evaluation. Additionally, we assume multiple (i.e., at least two) clouds each running a server (in the following called *cloud server*). In order to perform a secure computation, clients and cloud servers perform the following subprotocols:

(i) Each client runs the *input* protocol with the cloud servers to provide her private input in an MPC-specific encrypted form.

(ii) The cloud servers run the *main* MPC protocol and store the results of the program (still in encrypted form). Roughly speaking, the cloud servers together evaluate the desired function on encrypted data and do not get any information about plaintext inputs/outputs or intermediate results.

(iii) Each client runs the *output* protocol with the cloud servers to receive the private outputs assigned to her.

Apart from the better security guarantees, our approach has several further advantages compared to the direct approach:

- Lightweight clients suffice. The heavy computation happens in the main protocol run among the cloud servers. This protocol does not need any communication or interaction with the clients.
- The main protocol is independent of the number and set of clients that provide input or obtain output. Hence, this approach can run with a large and flexible set of clients.
- Clients do not need to be online all the time, only for providing input and receiving output. The mentioned phases of two-phase MPC protocols are carried out solely on the cloud server side.

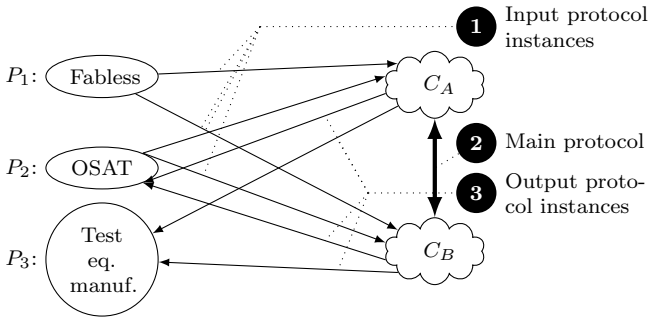

Figure 2: Cloud-based MPC involving three clients P_1, P_2, P_3 and two cloud servers C_A, C_B. P_1 and P_2 provide inputs; P_2 and P_3 obtain outputs.

- This approach benefits from the common advantages of cloud-based approaches, mentioned above (e.g., scalability and availability).

One disadvantage of this approach might be the costs involved in renting cloud resources.

3 Implementation of the Cloud-Based Approach

In this section, we present our implementation for cloud-based MPC, with applications to the semiconductor production chain presented in Section 4.

Current MPC frameworks, like the prominent MP-SPDZ [Kel20] and SCALE-MAMBA [ACC+21] frameworks, come with difficulties, especially in the context of cloud-based MPC. This includes the lack of an input and output protocol, missing support for storing secret values for reuse in future MPC executions, and no means to coordinate the tasks of the involved cloud servers. We propose solutions to all of these problems.

We have implemented input and output protocols based on (theoretical) protocols proposed by Damgård et al. [DDN+16]. These protocols are very lightweight and do not induce any significant performance overhead.

The issue of storing secret values, is critical i) for storing encrypted inputs from the input protocol and ii) for splitting computations. E.g. it allows to perform parts of a computation already before all inputs are available. Furthermore, with this ability it is possible to reuse the same secret value in multiple computations, reducing the total computation time. We extend the MP-SPDZ framework to be able to store secret values using a Cassandra database.

We have also developed a toolkit that automates everything that happens in the cloud. Using our toolkit, cloud servers provide an API through which clients can initiate the necessary protocol runs (input, main, and output). Whenever possible, a client has to trigger only one cloud server. The cloud servers then automatically coordinate their actions autonomously.

We note that while we use the term "cloud server", it not necessarily corresponds to a single (virtual) machine. Instead, in our toolkit, each cloud server has the form of a Kubernetes (K8s) cluster. K8s is the de-facto standard container orchestration system in cloud computing. A K8s cluster can span as many as 5000 machines [The21] which allows for running highly parallel computations on large datasets such as those from semiconductor testing.

3.1 Security and Privacy Guarantees

Our toolkit provides strong *correctness* and *privacy* guarantees, with so-called active security. More specifically, the protocols come with the following guarantees:

Correctness. Even if only one cloud server is honest the clients are guaranteed to obtain the correct output. That is, the servers cannot manipulate the computation without being detected

by the honest server, who would then abort the protocol.

Privacy. Even if all but one cloud server are malicious, no client and no cloud server learns any information about any inputs or outputs of other parties, or about intermediate computation results. The only thing clients can learn is the output they obtain (and whatever they can deduce from it).

Privacy and Correctness carry over from the MPC protocols that we use, namely the SPDZ protocols [DPSZ12, KOS16, KPR18] as implemented in the MP-SPDZ framework as well as the input and output protocols by Damgård et al. [DDN+16] that we implemented.

Privacy can be analyzed protocol by protocol: In the input protocol the servers only obtain *shares* of the inputs. The used full-threshold additive secret sharing scheme then guarantees that as long as one server remains honest and keeps her shares secret, the others are not able to deduce any information about the inputs themselves. In the main protocol all opened values are masked with one-time pads and therefore look random to everyone. The randomness of the masks is guaranteed by the privacy properties of the offline phase. Finally in the output protocol the actual result can only be reconstructed by the client since the shares of the honest server remain hidden from all other servers.

For correctness note that the input phase from [DDN+16] guarantees that the servers receive correctly authenticated shares of the inputs if all servers follow the protocol. If a malicious server deviates from the protocol, the shares are incorrectly authenticated with overwhelming probability. In the latter case the MAC check at the end of the main protocol will fail with overwhelming probability. The main protocol [DPSZ12, KOS16, KPR18] furthermore guarantees that malicious behavior during the execution of the main protocol itself will be caught in this MAC check, too. Finally, the output protocol comes with one server-side check and one client-side check that (if successful) guarantee that the client receives the correct result with overwhelming probability.

We note that for the security properties to hold true it does not matter how many clients are malicious. We also point out that the security properties do not only protect against a specific attack by an adversary but rather guarantee privacy and correctness (with abort) for every form of misbehavior of malicious parties (cloud servers or clients).

4 Application to Privacy-Preserving Semiconductor Testing

In this section, we evaluate our toolkit for cloud-based MPC by applying it to selected algorithms from semiconductor testing. Testing occurs in many stages of the production chain. For our case-study we used data from wafer acceptance tests (WATs) and wafer sort (WS) from a real-world ASIC production line by Advantest. In both cases, the data has the form $\mathbf{D} \in \mathbb{R}^{n_{\mathbf{rows}} \times n_{\mathbf{cols}}}$ where each row corresponds to a wafer (in the case of WATs) or die (in the case of WS) and each column corresponds to a measuring point. Each cell of this matrix contains the measured value at the specific measuring point of the specific wafer or die.

4.1 Example Applications

In order to evaluate our toolkit, we considered two algorithms from the example scenario (Section 2.1) of test execution and monitoring (namely, threshold counting and process capability index) and one algorithm that solves a problem (namely, detecting outliers) in the scenario of equipment health monitoring.

Threshold Counting. Given data $\mathbf{D} \in \mathbb{R}^{n_{\mathbf{rows}} \times n_{\mathbf{cols}}}$, lower thresholds $\mathbf{l} \in \mathbb{R}^{n_{\mathbf{cols}}}$ and upper thresholds $\mathbf{u} \in \mathbb{R}^{n_{\mathbf{cols}}}$ the task is to count, for each row, how many values exceed a threshold.

That is, for each row i, we need to compute $\sum_j 1_{[\mathbf{D}[i,j] \notin [\mathbf{l}[j], \mathbf{u}[j]]]}$. A wafer/die that adheres to the specification is characterized by a sum of zero.

The straightforward algorithm that computes this count has one counter for each row and proceeds as follows. For each matrix entry $\mathbf{D}[i,j]$, check if $\mathbf{l}[j] \leq \mathbf{D}[i,j] \leq \mathbf{u}[j]$. If the check fails, add 1 to the counter of the corresponding row. However, note that in MPC we can not directly evaluate an if-branch in a way that reveals which branch was taken because that would reveal to the players the result of the branching condition. Therefore we must write the algorithm in a branchless way by computing the condition as a boolean $\in \{0,1\}$ and adding it to the count.

The algorithm requires around $2n_{\mathbf{rows}}n_{\mathbf{cols}}$ comparisons and additions (for the increments). The bottleneck clearly are the comparisons. Not only does their computation require inter-party communication, but it also consumes preprocessed data (14.88 kB per player per comparison in our setup—see Section 4.2).

Detecting Outliers. Given data $\mathbf{D} \in \mathbb{R}^{n_{\mathbf{rows}} \times n_{\mathbf{cols}}}$ and under the assumption that each column is normally distributed except for some potential outliers, the goal is to detect these outliers. This can be done using the Grubbs test [Gru50]. We implemented this test as an MPC program GrubbsTest which requires around $2n_{\mathbf{rows}}n_{\mathbf{cols}}$ comparisons, additions and multiplications.

Process Capability Index. In our final example, given data $\mathbf{D} \in \mathbb{R}^{n_{\mathbf{rows}} \times n_{\mathbf{cols}}}$, lower thresholds $\mathbf{l} \in \mathbb{R}^{n_{\mathbf{cols}}}$ and upper thresholds $\mathbf{u} \in \mathbb{R}^{n_{\mathbf{cols}}}$ the goal is to compute, for each column, the so-called process capability index ($\hat{C}_{pk}$) [oST12] and check whether it is sufficiently high. We implemented this test as an MPC program CpkTest. It requires around $2n_{\mathbf{rows}}n_{\mathbf{cols}}$ multiplications and additions, and only one comparison at the end.

4.2 Evaluation

In this section, we evaluate our toolkit with regards to the example algorithms above. We discuss the performance, (vertical and horizontal) scalability, and cost of running them on WAT and WS test data from a real-world ASIC production line that has been kindly provided by Advantest. We use two cloud servers, each being a managed K8s clusters from Amazon's Elastic Kubernetes Service (EKS) [Ama22b], using t3.medium [Ama22a] instances as worker nodes. These instances are VMs with 2 vCPUs at 2.5 GHz, 4 GiB RAM und up to 5 Gbit/s network bandwidth. When running benchmarks with multiple parallel MPC executions, we scaled the clusters to sufficient nodes such that each MPC player can run on a separate instance.

Performance. Recall that MASCOT (as well as other SPDZ-based MPC protocols) consists of an offline phase followed by an online phase. Only the online phase is input-dependent. Therefore, we can compute the offline phase ahead of time. When doing so, the response time (i.e., the time from when we provide the inputs to when we obtain the outputs of the computation) is entirely given by the time required for the online phase. In Table 1 we show the time and network traffic required for both the offline and online phase. The times are given per sample. We took averages from running the presented algorithms in our environment 10 times each.

The run time for our example applications—theoretically and empirically—scales linearly with the number of samples. However, if a fast response time is desired, then this might not be fast enough. To speed up the computatoin, one can utilize horizontal scaling, which means that more machines are added to the computation. It is the preferred way of scaling in cloud environments and is well supported by K8s. In order to utilize multiple machines using our cloud-based MPC toolkit, one simply starts multiple MPC computations simultaneously. K8s automatically schedules the players to machines with sufficient free resources. In order to horizontally scale the execution of a single computation, the clients need to split it up into multiple independently running computations. We successfully ran as many as 83 parallel instances

Table 1: Offline vs. Online Phase

Algorithm	Offline		Online	
	Time[a]	***Traffic***[a]	***Time***[a]	***Traffic***[a]
TCounting	389 ms	20.3 MB	10.0 ms	15.4 kB
GrubbsTest	390 ms	20.5 MB	13.2 ms	15.6 kB
CpkTest	1.6 ms	63.2 kB	0.15 ms	68.8 B

[a]per sample.

of TCounting, scheduled to 83 different machines per cloud server, and achieved the same per-instance performance as when running only one instance.

Cost. When running our example applications in our environment, about 98 percent of the cloud computing costs incure due to network traffic. Note that most of the network traffic is produced by the offline phase. Therefore, the main way to reduce cost of secret-sharing-based MPC is to optimize the offline phase for less network communication. This is relevant not only for the main protocol, but also for the input and output protocol, since they also require preprocessed data from an offline phase.

5 Offline Phase with Reduced Communication

In our evaluation (Section 4.2) we used a variant of SPDZ called SPD$\mathbb{Z}_{2^k}$ [CDE+18] which performs computations over the domain $\mathbb{Z}_{2^k}$, as this has the fastest online phase and thus the fastest response time. Currently, the only offline phase implemented for SPD$\mathbb{Z}_{2^k}$ is based on MASCOT and has a quite high communication requirement, thus leading to large costs in the cloud.

To overcome this problem, we revisit a HE-based offline phase called Overdrive [KPR18]. Overdrive comes in two variants: the LowGear variant that is most efficient for a low number of parties and the HighGear variant which has a better asymptotic complexity in the number of parties. In [OSV20] it was shown that the Overdrive protocols can be adapted for MPC over $\mathbb{Z}_{2^k}$. This approach is called Overdrive2k and was later improved in [CKL21], but the approach was never implemented in software, neither was *any* HE-based offline phase for $\mathbb{Z}_{2^k}$.

While previous works on Overdrive2k [OSV20, CKL21] focus on the HighGear variant, we focus on a low number of parties, as this is typically the case in the cloud-based approach. E.g. it is quite common to use just two cloud servers. We have an ongoing work where, for the first time, we take a closer look at the LowGear variant of Overdrive2k and present several optimizations for reducing its communication requirement. In our protocol, which we call Multipars, the amortized communication is more than 2x better than the previous best [CKL21] in the two-party setting. In particular, this ongoing work contains the following contributions:

- We transfer a recent approach [RRKK22], that eliminates the sacrifice step in LowGear, to the $\mathbb{Z}_{2^k}$ setting. We show that this is insecure when using the Overdrive2k packing technique [OSV20] where an adversary could construct a plaintext that doesn't correspond to a valid message. We solve this problem by instead using the *tweaked interpolation packing* from [CKL21] that comes with a zero-knowledge proof of message knowledge. Furthermore, for ciphertexts where pairwise multiplication is not required, we use the (much more efficient) coefficient packing together with smaller BGV parameters.

- We employ modulus switching [BGV12] to reduce the size of the majority of ciphertexts that need to be transmitted. While modulus switching is traditionally used to reduce the noise

Table 2: Amortized communication in kbit per triple produced with $N = 2$ parties

k	s	SPD$\mathbb{Z}_{2^k}$ [CDE+18]	Mon$\mathbb{Z}_{2^k}$a [CRFG20]	Overdrive2k [OSV20]	MHz2k-TG2k [CKL21]		Multipars	
					$U = 2V$	$U = 4V$	$U = 2V$	$U = 4V$
32	32	79.87	59.07	101.8	26.4	20.1	≈ 11	≈ 9
64	64	319.49	175.49	171.4	43.3	31.9	≈ 16	≈ 13
128	64	557.06	176.64	190.4	55.0	40.9	≈ 21	≈ 17

of ciphertexts, it can be used for our purpose, too, reducing overall communication by about 50%.

- We provide security against maliciously generated HE keys, whereas previous works [DPSZ12, KPR18, OSV20, CKL21] assume a trusted dealer that generates and distributes the keys. We construct a secure key generation protocol where the owner proves knowledge of the secret key using a zero-knowledge proof of secret key knowledge. This introduces a certain slack on the secret key, which is taken into account by our security analysis of Multipars.
- We discovered several minor technical optimizations related to the zero-knowledge proofs, security analysis, and software implementation.
- We implemented Multipars as a Rust software, thus providing the first-ever software implementation of a HE-based offline phase for SPD$\mathbb{Z}_{2^k}$.

In Table 2 we compare our work to prior offline phases for MPC over $\mathbb{Z}_{2^k}$. For Overdrive2k, we include the numbers from [CKL21], as the communication costs originally given in [OSV20] are hard to reproduce [CKL21]. The numbers for Multipars are approximate, as our work is still ongoing. Our comparison shows that Multipars outperforms the SPD$\mathbb{Z}_{2^k}$ offline phase implemented in MP-SPDZ [DEF+19], which is based on MASCOT [KOS16], by a factor of 8–32x, depending on domain and security parameters.

We are in the process of evaluating how this saving in communication translates to concrete cost savings when using our new offline phase for privacy-preserving computations in the semiconductor production chain.

6 Summary and Future Directions

We analyzed security setups that regularly occur in the production process and propose a privacy-preserving solution based on a secure two-phase MPC protocol. Our secure protocol integrates the state-of-the-art MPC protocols included in [Kel20] in a cloud infrastructure to deal with data processed in the semiconductor production chain. We implemented a newly created toolkit and evaluated its performance on real-world data from the semiconductor industry. The resulting benchmarks show that standard operations needed in the semiconductor testing process can be computed efficiently in a privacy-preserving way. Obviously, the performance is not comparable with computations on plain data. But our approach comes with very high security and privacy guarantees. We therefore see our research as a first promising demonstration that the MPC approach in general and our implementation in particular can successfully be employed to enable privacy-preserving semiconductor production. Clearly, this work should be considered to be a starting point of this exciting new research field. Many open questions remain.

The MPC approach will be further improved, benchmarked, and applied to more and bigger scenarios. For instance, we're already working on vastly reducing network traffic in the offline phase in order to reduce cost. Furthermore, for a setting where computations must run on the test floor instead of in the cloud, an on-premises approach involving ACS Edge was started to be discussed.

References

[ABL+18] David Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Pagter, Nigel Smart, and Rebecca Wright. From Keys to Databases. *Computer Journal*, 61:1749–1771, 2018.

[ACC+21] A. Aly, K. Cong, D. Cozzo, M. Keller, E. Orsini, D. Rotaru, O. Scherer, P. Scholl, N.P. Smart, T. Tanguy, and T. Wood. SCALE–MAMBA v1.14 : Documentation, May 2021.

[Als21] Thomas Alsop. Fabless/system company and integrated device manufacturer IC sales 1999-2020, 2021.

[Ama22a] Amazon Web Services Inc. Amazon EC2 T3 Instances, 2022.

[Ama22b] Amazon Web Services Inc. Amazon Elastic Kubernetes Service, 2022.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *ACM ITCS '12*, page 309–325, 2012.

[CDE+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spd$\mathbb{F}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In *CRYPTO 2018*, pages 769–798. Springer, 2018.

[CKL21] Jung Hee Cheon, Dongwoo Kim, and Keewoo Lee. Mhz2k: MPC from HE over $\mathbb {Z}_{2^k}$ with new packing, simpler reshare, and better ZKP. In *CRYPTO 2021*, pages 426–456. Springer, 2021.

[CKR+20] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning. In *ASIACRYPT '20*, pages 31–59. Springer, 2020.

[CPR19] Valerie Chen, Valerio Pastro, and Mariana Raykova. Secure Computation for Machine Learning With SPDZ. 2019.

[CRFG20] 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 *PKC 2020*, pages 357–386. Springer, 2020.

[DDN+16] Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential Benchmarking Based on Multiparty Computation. In *Financial Cryptography and Data Security '16*, pages 169–187. Springer, 2016.

[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 *IEEE S&P '19*, pages 1102–1120, 2019.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO '12*, pages 643–662. Springer, 2012.

[DSFDNR19] Mathieu Da Silva, Marie-Lise Flottes, Giorgio Di Natale, and Bruno Rouzeyre. Preventing Scan Attacks on Secure Circuits Through Scan Chain Encryption. *IEEE TCAD*, 38(3):538–550, 2019.

[Dwo06] Cynthia Dwork. Differential Privacy. In *Automata, Languages and Programming*, pages 1–12. Springer, 2006.

[Gen09] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *ACM STOC '09*, pages 169–178, 2009.

[Gru50] Frank E Grubbs. Sample Criteria for Testing Outlying Observations. *The Annals of Mathematical Statistics*, 21(1):27–58, 1950.

[HCW17] Hui-Chih Hung, Yu-Chih Chiu, and Muh-Cherng Wu. Analysis of competition between idm and fabless–foundry business models in the semiconductor industry. *IEEE Trans. Semicond. Manuf.*, 30(3):254–260, 2017.

[IEE20] IEEE. International Roadmap for Devices and Systems - Executive Summary, 2020.

[JKK+19] Sung-Ju Jang, Jong-Seong Kim, Tae-Woo Kim, Hyun-Jin Lee, and Seungbum Ko. A Wafer Map Yield Prediction Basd on Machine Learning for Productivity Enhancement. *IEEE Trans. Semicond. Manuf.*, 32(4):400–407, 2019.

[Kel20] Marcel Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *ACM CCS '20*, pages 1575–1590, 2020.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *ACM CCS '16*, pages 830–842, 2016.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ Great Again. In *EUROCRYPT '18*, pages 158–189. Springer, 2018.

[KRC+20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow Inference. In *IEEE S&P '20*, pages 336–353, 2020.

[oST12] National Institute of Standards and Technology. What is process capability?, 2012.

[OSV20] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure MPC over $\mathbb {Z}_{2^k}$ from somewhat homomorphic encryption. In *CT-RSA 2020*, pages 254–283. Springer, 2020.

[RK10] Kurt Rosenfeld and Ramesh Karri. Attacks and Defenses for JTAG. *IEEE Design & Test of Computers*, 27(1):36–47, 2010.

[RRK18] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. DeepSecure: Scalable Provably-Secure Deep Learning. In *ACM DAC '18*, pages 2:1–2:6, 2018.

[RRKK22] Pascal Reisert, Marc Rivinius, Toomas Krips, and Ralf Küsters. Overdrive LowGear 2.0: Reduced-bandwidth MPC without sacrifice (under submission), 2022.

[Sch16] Matthias Schunter. Intel Software Guard Extensions: Introduction and Open Research Challenges. In *ACM Workshop on Software PROtection '16*, 2016.

[SKWT17] Mehdi Sadi, Sukeshwar Kannan, LeRoy Winemberg, and Mark Tehranipoor. SoC Speed Binning Using Machine Learning and On-Chip Slack Sensors. *IEEE TCAD*, 36(5):842–854, 2017.

[The21] The Kubernetes Authors. Considerations for large clusters, March 2021.

[YF19] Yang Yuan-Fu. A Deep Learning Model for Identification of Defect Patterns in Semiconductor Wafer Map. In *ASMC '19*, 2019.

[ZPGS19] Wenting Zheng, Raluca Popa, Joseph Gonzalez, and Ion Stoica. Helen: Maliciously Secure Coopetitive Learning for Linear Models. In *IEEE S&P '19*, pages 724–738, 2019.