# Secure Multi-Party Computation with SCALE-MAMBA

Jonas Rülfing

Seminar: Privacy and Big Data - RWTH Aachen

**Abstract.** This paper examines Secure Multi-Party Computation (SMPC), especially the SCALE-MAMBA protocol. First it introduces Secure Multi-Party Computation and fundamental primitives, and then takes a detailed look at SCALE-MAMBA, especially in comparison to its predecessor SPDZ. It focuses on the algorithmic foundations, especially the split into online and offline phase, how beaver triples are used to enable computational efficient multiplications and which computational complexities the algorithms posses. Afterwards, it tries to depict the evolution in the different versions of the protocol and tries to locate SCALE-MAMBA between its competitors. Ultimately, it will show some exemplary implementations in real world applications and will give an outlook what further evolutions might be coming in the next years.

## 1 Motivation and Introduction

In today's world, and even more in the future, there exist a lot of computations that have to be done in a distributed environment, because in a lot of use cases multiple parties are involved, but also because the amount of data is getting so big that it often becomes impossible to have a single entity that processes and stores all of it. Furthermore, in a lot of these cases the parties participating in these computations do not want to reveal their own information, because it is sensitive or it has a high business value.

A basic example is an anonymous voting. In most elections there is a third party that has to aggregate and count the votes. For instance in Germany's elections, this is done by specific persons counting the local votes then forwarding the information to a central institution. The other problem is that it has to be guaranteed that a vote is valid, meaning that each voter holds the right to vote and is only voting one time. So in an ideal system, no person participating in the aggregation process should be able to get any information over any vote or any subset of voters.

Another use case is the exchange of medical data. Medical data of patients is highly confidential and should never be published. But at the same time, a lot of different hospitals or doctors have only the data from their small data set of local patients, and for making founded diagnosis for new patients it would be helpful to compare the symptoms to the diagnoses of other doctors.

Secure Multi-Party Computation (SMPC) is one possible solution for these use cases, because we can model above mentioned and a lot similar problems as functions. SMPC is a field of research that emerged within the last 30 years, and is gathering a lot of speed in the last years. The basic problem is the evaluation of a common function, to which multiple parties have different inputs, without revealing these inputs. Although there are a lot of theoretical solutions for this problem, real-world implementations are still not performing on the desired level. So in the last years the research shifted more into the area of how to improve the performance of existing solutions while still guaranteeing a desired level of security.

This paper's main focus will lay on SCALE-MAMBA [1], a system developed mainly by the Katholieke Universiteit Leuven. This system is one of the forerunners in this area and combines a lot of research by different teams to provide a platform with highly performant SMPC. Nevertheless, this paper will at first give a short introduction into the history of SMPC and will try to explain its basic technologies to the reader. It will compare SCALE-MAMBA to its predecessors and will also try to give a short comparison to other SMPC systems which use different approaches, especially taking a look on which different requirements are fulfilled by each. Ultimately, it will try to give an outlook what might be coming in the next years.

## 2 SMPC Basics and History

### 2.1 Beginnings

As we saw in the introduction, there are a lot current, and much more upcoming, use cases of calculating the output of a function in a distributed environment with multiple parties while at the same time keeping all the individual inputs secret. The first explorations in these fields started in the 1970s, with theoretical problems like the mental poker problem [16], that asks how two players can play poker while not being in the same physical location and making sure no one is cheating, and Yao's millionaire problem [26], which tries to answer the question which of two millionaires is richer than the other while not revealing their respective wealth.

These explorations led to a group of basic protocols which established the area of Secure Multi-Party Computation(SMPC).

### 2.2 Problem statement

Formally, the SMPC problem can be stated as follows:

- Given a finite field $\mathcal{F}_p = \{0, ..., p-1\}$ with a prime p
- Given $n$ parties $P_1, ..., P_n$ participating in the computation
- Given $m$ secret inputs $x_1, ..., x_m$ with each of them hold by one of the parties
- Given a function $f(x_1, ..., x_m)$
- Goal: Evaluation of the function $f(x_1, ..., x_m)$ while guaranteeing secrecy and correctness

**Secrecy:** If party $P_i$ holds secret $x_j$, then at the end of the evaluation no information regarding $x_j$ is revealed to the other parties.
**Correctness:** At the end of the computation, the result is correct.

In a lot of cases each party has exactly one corresponding secret input, but this is not necessarily required. There can also be parties without input and parties with multiple inputs.

Now we will take a short look on two of the first protocols, which both are the basis for a whole family of SMPC schemes.

### 2.3 Yao's Garbled Circuit

There are typically two families, into which most of the SMPC protocols can be categorized. One is based on garbled circuits which were first introduced by Yao in 1986 [25]. The basic idea is that, in a two party setting, one party, Alice, encrypts the Boolean circuit that has to be evaluated and sends it to the second party, Bob. Bob uses his own input, Alice's encrypted input and the garbled circuit to evaluate the circuit in an encrypted way. At the end they exchange the results and can compute the result this way.

### 2.4 Shamirs Secret Sharing Scheme

This paper will not further discuss the family of garbled circuit protocols, but rather the second family, which are protocols based on so called Linear Secret Sharing. These type of protocols were introduced by Adi Shamir at a similar time, in 1979 [23].

The basic idea of this approach is the the so called share-compute-reveal paradigm, which splits the evaluation of the function into three parts.

**Share:** First, the inputs secrets are split into multiple parts which then are shared with the other parties. Hence, every party holds a share of each input secret.
**Compute:** The parties evaluate the function with the secret input shares. The protocol has to be defined in a way, that the final result which each parties computes with the input shares is a respective share of the complete final result.
**Reveal:** The parties aggregate their shares of the final result and compute the final result together.

This paradigm already shows a lot of challenges arising for a concrete protocol, for instance:

- How can the secrets be shared in a homomorphic way, so that, after computing on shared inputs, the complete result can be recovered from the result shares?
- While this can be done for addition in easy ways, how can this be done for multiplication as well?
- How can the correctness of the computation be verified?

- How can the overhead generated by a protocol can be kept on a reasonable scale, so that a protocol is applicable to a real-world use case?

In Shamir's scheme the secret sharing is implemented by expressing a secret $s$ and its shares as a random polynomial $f(x)$, so that $f(0) = s$ and each share $s_i$ has the value $s_i = f(i)$. Computations are executed on these shares, and at the end it is possible to reconstruct the result from the different shares without revealing information about the initial secrets. This is done by using Lagrange interpolation. The degree $p$ of the polynomial can be seen as a security parameter, because $p + 1$ shares are needed to reconstruct the secret.

There is a huge family of protocols build on this approach, and it is still evolving every day. This paper will focus on the newest of these protocols that were developed in the last 10 years, especially on the protocols which are based on or similar to SPDZ [10] which was published in 2011.

Before looking into exact implementations of linear secret sharing schemes, following, a short look will taken at the general adversary model and required basic definitions.

### 2.5 Adversary Model

In cryptography, the adversary which is considered is normally a third party that wants to get information for his own gain or want to sabotage a protocol because of his own interest. In SMPC it is expected that the adversary is not only participating as one party, but could control multiple parties. That means a potential adversary can control between one and all but one of the parties which are participating. The difference to the normal cryptographic adversary model is, that the adversary is controlling parties that are participating in the scheme and not a third party that tries to intercept communication.

In the area of SMPC there is normally a differentiation between two types of adversaries.

**Honest-but-curious** The honest-but-curious adversary tries to gain as much information as possible while obeying the rules of the protocol. Hence, it is impossible to know that this party is controlled by a third entity. This adversary tries to retrieve some of the secret information from the other parties, by using all the information he gets during the execution of the protocol.

**Malicious adversary** While this adversary also wants to gain all possible information, he is not obeying the rules of the protocol. This type of adversary can send different inputs to manipulate the results he receives. Furthermore, he wants to interrupt the whole protocol. That could mean making the computation impossible, but as well modifying his own input or his own calculations, so the final result is changed and might lead to different implications.

It is important to distinguish between these two types of adversary, because different measures are needed for each adversary. Additionally, some protocols only provide passive security, which means they protect against a honest-but-curious adversary but not against a malicious one.

### 2.6 Access structures

There are a lot of different use cases for SMPC, so of course the adversary model changes depending on the system, the security needs, and the amount of parties involved. It is important to differentiate between these different use cases, because the different needs have big implications for security measures that have to be taken and, thus, for run-time complexity. SMPC should run as fast as possible, while still providing all the necessary security guarantees.

To describe which party has which rights, and how many parties can be corrupted, so that the security can still be guaranteed, access structures were introduced. An access structure describes how many parties are needed to form a so called qualified set. A qualified set is a group of parties which can pool their information to extract secret information. An access structure is called $Q_l$-access structure if at least $l+1$ parties are needed to form a qualified set. An $(n,t)$-threshold scheme with $t < \frac{n}{l}$ means that at least $t+1$ parties are needed to form a qualified set [9]. Here, $n$ is the number of total parties, and $t$ is the number of possible adversaries while not revealing any secrets.

A full threshold scheme means that all parties are needed to form a qualified set. This implies, that, if all other parties except one are corrupted by an adversary, this one party can still rely on the fact that its secret remains secret. Of course this has a huge impact on overhead and performance, but in some cases it can provide needed security guarantees.

An access structure also does not have to be symmetric. In some cases there might be different type of parties, where different parties have different positions or levels of trust, so it is, for example, possible to specify that some set of parties only has to have the size of 2 to form a qualified set and another one needs the size of 4.

Choosing the right access structure for the right use case is important, because it has a big impact on performance and on the desired security level.

## 3 SCALE-MAMBA

### 3.1 Overview

In the last years, research in SMPC is constantly ongoing, and there are new approaches and improvements to old approaches appearing regularly. But most of these publications only focus on a small area of the whole picture. SPDZ [10] is a system that appeared in 2011 and, with a lot of evolutions, was one of the most refined SMPC schemes.

SCALE-MAMBA[1] is the newest successor of SPDZ and is a project that tries to combine most of the state-of-the-art research into one system. This implies that it is offering an end-to-end system for SMPC.

It is a research system that is continuously in development, therefore, there is not a single final version that can be examined. It offers a wide range of protocols for different use cases, i.e. it is possible to use the system in different configurations. That also means for this paper, that there is not a single protocol that can be examined. Thus, this paper tries to give more of a overview how the process works in general, which are the main, especially newest and best performing, components and where are the problems and bottlenecks. This overview is only selective and by no means complete.

SCALE-MAMBA consists of two parts. One part is called Secure Computation Algorithms from LEuven (SCALE) and is the SMPC part. It contains all the computational logic and algorithms for multi-party computations. The second part is called Multiparty AlgorithMs Basic Argot (MAMBA) and is a compiler for the SMPC programs.

In general, the SCALE protocol is split in two parts, first an offline phase and afterwards an online phase. In the online phase only the final evaluation of the function is executed. In the offline phase, prior to the online phase, preparations are done by all parties to prepare for the evaluation. This already includes calculating values for possible multiplications and message authentication codes, so that the online phase can be executed faster. In the offline phase the expensive calculations are done, like public key cryptography or authentications, so that the online phase only needs basic primitives. The offline phase is not completely offline, because the parties still exchange some information which are later needed for the evaluation. But this phase is not that time critically, that means it can be already executed a lot earlier than the online phase, so that, when the function evaluation is really needed, the result can be calculated in a rather fast time compared to a system that still has to set up everything. In SCALE-MAMBA, compared to its predecessors, the offline and online phase are completely integrated. Therefore, it is not possible anymore to execute them separately like it was in SPDZ for example. This is done to be closer to a real-world implementation.

While SCALE-MAMBA also offers different access structures, the most commonly used security configuration is active security with abort in a full threshold scheme, that means that every party except one could be corrupted, and still no information is leaked, because the evaluation is aborted.

### 3.2 MAMBA

For this paper the most interesting part is the algorithmic. But still, the importance of the compiler should not be undervalued. Although the algorithmic foundations might be more important from a theoretical point of view, the compiler, written in C++, enables the user to use these technologies in a system where he can rely on the security guarantees given by the algorithms. The programs can be written for MAMBA in a syntax similar to python's. In these programs, the user can specify which security settings he wishes to use. He can specify, how he wants his SMPC to exactly be executed, and which security parameters he wants to use. MAMBA compiles the instructions to bytecode which is then executed by SCALE. Because of MAMBA, it is much more easy to test and compare different configurations. This abstraction layer makes the system more usable and analyzable.

### 3.3 Secret sharing

As mentioned before, SCALE-MAMBA and similar protocols implement the share-compute-reveal paradigm. The sharing is implemented as follows:

Given a party $P_i$ which holds a secret value $a \in \mathcal{F}_p$ and a share of the global MAC key $\alpha$. Then $P_i$ can additvely share $a$ with the other $n-1$ parties as follows:

1. Choose random $\delta_a \in \mathcal{F}_p$
2. Compute tag $\gamma(a) = \alpha(a + \delta)$
3. Choose random $a_1, ..., a_n \in \mathcal{F}_p$ so that $a = (a_1 + ... + a_n)$
4. Choose random $(\gamma(a)_1 + ... + \gamma(a)_n) \in \mathcal{F}_p$ so that $\gamma(a) = ((\gamma(a)_1 + ... + \gamma(a)_n))$
5. Send $\delta_a$ and the respective shares $a_j$ and $\gamma(a)_j$ to each party $P_j$
6. Now each party $P_j$ holds a share of $a$ as tuple $\langle a_j \rangle = (\delta_a, a_j, \gamma(a)_j)$

Now the value of $a$ and the corresponding tag $\gamma(a)$ can only be reconstructed if, and only if, all $n$ secret shares are known. At the same time, no bit of information regarding $a$ is revealed, as long as not all secret shares are known.
**Correctness:** Assume that $n - 1$ secret shares are known to an adversary, i.e. all secret shares except one $a_x \in \mathcal{F}_p$. Then the adversary can calculate $\sum_{i \neq x} a_i = a - a_x$. Nevertheless, because $a_x$ was drawn from an independent distribution and the adversary has no information over the value of $a_x$, knowing $a - a_x$ reveals not a single bit of information about $a$.

### 3.4 Revealing secrets

SCALE-MAMBA uses two different types of revealing a secretly shared value.

**Partial Reveal** All players send their respective shares of a value, but not the tag share, to one chosen party $P_i$ which sums up all values and then broadcasts the result. The tag stays secretly shared between the parties.
In praxis, the party computing the value is changing for every reveal to keep the workload balanced.
**Full Reveal** In this case all players broadcast their respective shares and the respective MACs to all players, so all players can calculate the secret and verify the result with the corresponding tag.

### 3.5 Offline phase

The offline phase prepares everything, so the function can be evaluated as efficient as possible in the online phase. There are two main things done in the offline phase. The first one is the generation of a global Message Authentication Code(MAC) key $\alpha$. This MAC key is used later, to authenticate the different shares at the end of the computation.

The second and and most important part in the offline phase, is the generation of the beaver triples [4]. These triples are needed in the online phase to express multiplications as a combination of additions and multiplications with constants. For each multiplication, one of the triples is needed.

**Generation of the MAC key** Message Authentification Codes (MACs) are used to guarantee the integrity and authenticity of a message. Basically, a MAC is a key that is used to generate a tag for a message. This tag is used to verify the identity of the sender and that the content of the message was not changed. For using MACs we need a tupel of two algorithms $(\text{MAC}, \text{VER})$:

1. The MAC algorithm $\text{MAC}(K, m) \to t$ takes a key $K$ and a message $m$ and outputs a tag $t$
2. The verficiation algorithm $\text{VER}(K, m, t) \to 0/1$ takes a key $K$, a message $m$ and a tag $t$ and outputs accept(1) if the tag is correct or reject(0)
   if it is not

In SCALE-MAMBA really simple MACs are used. We assume that we have a MAC key $\alpha$. Then the MAC algorithm used is following:

$$\text{MAC}(\alpha, m) \to \alpha * m$$
$$\text{VER}(\alpha, m, t) \to (1 \Leftrightarrow t = \alpha * m)$$

Compared to other MAC algorithms, this algorithm has high storage complexity because the tag has the same size as the message. But it has other advantages, which are that it is not computation complex with only one multiplication, it is symmetric and it is homomorphic.

The MAC key is distributively generated, so that every party has a additive share $\alpha_i$. Only at the end of the computation, the parties aggregate their shares to verify the result.

**Beaver Triples** Beaver triples, introduced in [4], are triples of three numbers $(a, b, c)$, with $a, b, c \in \mathcal{F}_p$, with $a$ and $b$ being chosen uniformly randomly, and $a * b = c$. In the offline or pre-processing phase, the most important part is the generation of the beaver triples. These triples are independent of the computation that is later performed in the online phase, but will be used in it to guarantee that multiplications can be done without revealing the secret inputs of the different parties.

For each multiplication operation in the online phase, we need one of the Beaver triples $(\langle x_1 \rangle, \langle y_1 \rangle, \langle z_1 \rangle)$ generated in the offline phase. In the offline phase we checked the quality of our triples, but we still allowed the possibility that each triple has an error $e$ so that $c = a * b + e$. To check the quality of our triple, we use a second triple $(\langle x_2 \rangle, \langle y_2 \rangle, \langle z_2 \rangle)$ in a procedure called sacrificing it. Both triples are additively shared between all parties like the input values. In SPDZ both triples are completely unrelated to each other, and the sacrificing is done as follows:

For the generation of a beaver triple $(a, b, c)$, homomorphic public key cryptography based on the Brakerski-Gentry-Vaikuntanathan encryption scheme [6] is used. Every party $P_i$ generates for the communication with each party $P_j$ a set of keys $(pk_{ij}, sk_{ij})$. Then, one triple is generated as follows:

1. Every party $P_i$ samples a random $a_i$ and $b_i$; then $a = \sum_{j=1}^{n} a_i$ and $b = \sum_{j=1}^{n} b_i$
2. Every party $P_i$ executes following protocol with each other party $P_j$:
   (a) $P_i$ sends $P_j$ $Enc_{pk_{ij}}(a_i)$
   (b) $P_j$ samples random $e_{ij}$, calculates $g_{ij} = b_j * Enc_{pk_{ij}}(a_i) - Enc_{pk_{ij}}(e_{ij})$ and sends $g_{ij}$ it to $P_i$
   (c) $P_i$ decrypts $d_{ij} = Dec_{sk_{ij}}(g_{ij})$
3. Every party $P_i$ computes $c_i = a_i * b_i + \sum_{i \neq j}(e_{ij} + d_{ij})$

Now, $c$ is correctly shared so that $c = c_1 + ... + c_n$.
**Correctness:**

$$d_{ij} = b_j * a_i - e_{ij}$$

$$c_i = a_i * b_i + \sum_{i \neq j}(e_{ij} + d_{ij}) = a_i * b_i + \sum_{i \neq j}(b_j * a_i)$$

$$c_i = a_i * b_i + a_i * \sum_{i \neq j}(b_j) = a_i * (b_i + \sum_{i \neq j} b_j) = a_i * b$$

$$c = c_1 + ... + c_n = a_1 * b + ... + a_n * b = (a_1 + ... + a_n) * b = a * b$$

**Zero-Knowledge Proofs** In the many iterations of the SPDZ like protocols, there were different ways to guarantee that the parties are not cheating. An important technology for that are the so called Zero-Knowledge Proofs(ZKP). Here, they are used to guarantee that the Beaver triples are not diverting more than a specified error $e$ from the expected value. ZKPs are a technique that was introduced in 1985 in [15]. The idea is that one party can prove to another party that it knows a specific value $x$, without revealing the value of $x$ to the other person, nor revealing any other knowledge about the value.

Assumed Alice, who is called the prover, wants to prove to Bob, the verifier, that she knows $x$ in a Field $\mathcal{F}$, such that $f(x) = y$. It is as well assumed, that $f$ is homomorphic with respect to the field operations in $\mathcal{F}$.

1. Alice chooses a random $s \in \mathcal{F}$ , calculates $a = f(s)$ and sends $a$ to Bob
2. Bob samples a random $e$ out of $\mathcal{F}$ and sends it to Alice
3. Alice calculates $z = s + e * x$ and sends it to Bob
4. Bob checks if $f(z) = a + e * y$

**Correctness:** Bob wants to prove that $z$ provided by Alice contains the proper $x$, so that $f(x) = y$. Because f is homomorphic, obviously $f(z) = f(s) + f(e * x) = f(s) + e * f(x) = a + e * y$ with $f(s) = a$ and $f(x) = y$. So Bob has to calculate $a + e * y$ and $f(z)$ with the provided $z$ by Alice. If they are equal, he knows that

Alice knows the $x$ so that $f(x) = y$. He can do that, because he got provided the $a$ and $z$ by Alice, he knew the $y$ before the protocol and he chose the $e$ himself.
**Confidentiality:** We also have to make sure, that Alice did not share any information regarding her x. In the first step she chose a random element s and calculated a=f(s). This information is independent on her secret x, so she did not share any information. When calculating z=s+e*x, she also did not share any information regarding x, as long it is guaranteed that Bob did not have any knowledge about the s. So under the assumption that Alice chooses a new random s each time the protocol is executed, she doesn't share any information regarding her secret x.

In SCALE-MAMBA a variant of ZKPs similar to Schnorr's protocol [22] is used, but this paper will not examine the exact protocol.

**First improvements** In one of the earlier works, BDOZ [5], pairwise MACs were used to authenticate the secret sharing between the parties. It used a pairwise multiplication protocol with linear homomorphic encryption. To guarantee the data integrity, pairwise zero-knowledge proofs were used, so in total $O(n^2)$ of them, with $n$ being the numbers of parties. Because Beaver triples are created secretly shared, this accumulated a high overall run time because a lot of these triples are needed.

In SPDZ, semi homomorphic encryption based on the Brakerski-Gentry-Vaikuntanathan cryptosystem [6] was used. Still pairwise ZKPs were used, but one proof can be made to prove thousands of triples at once. This reduced the amount of proofs needed by a linear factor.

Nevertheless, the offline phase was still the weak point of the existing SPDZ implementations regarding run time complexity. In 2016 MASCOT [17] was introduced, which improved the speed of the offline phase by a magnitude of 2. MASCOT used Oblivious Transfer instead of SHE to significantly reduce the amount of communication and computation. We will not go into detail into this protocol, but rather look on its successor.

Because the offline phase was the weak point of the existing SPDZ implementations, in regards to run time while still managing a sufficiently secure system, in 2016 MASCOT [17] was introduced, which improved the speed of the offline phase by a magnitude of 2. MASCOT used oblivious transfer instead of SHE to significantly reduce the amount of communication and computation. We will not go into detail into this protocol, but rather look on its successor, which is really similar.

**Overdrive** Shortly afterwards in 2017 a new protocol for the offline phase called Overdrive was introduced in [18], which once again used semi homomorphic encryption instead of oblivious transfer. It was able to further improve the run time of the offline phase of MASCOT by a factor of 6. Overdrive uses the same ideas that were introduced in MASCOT, but applies them to semi homomorphic encryption. It includes two parts, a protocol called Low Gear for small numbers, and a protocol called High Gear for larger numbers. In version 1.2 of Scale-mamba only the HighGear protocol was used, even for smaller numbers.

The biggest difference between Overdrive and SPDZ is, that the parties not anymore prove multiple smaller statements between each other with ZKPoK, but instead make a joint statement and prove this one toegether. Because ZKPs are computational expensive, this makes a big difference in the overall performance.

This way, Overdrive did not improve the amount of communication channels used, but instead it decreased the computational costs by a linear factor.

The zero proofs of knowledge allow a so called soundness slack [8]. The idea is that the proof allows a small error in the value, because it is cheaper to prove than to prove the exact value. This error is later eliminated by a technique called sacrificing the beaver triples.

Additionally, drowning is used in Overdrive, which is the process of adding random noise to a secret sharing so the encryption does not reveal any information over the secret.

**TopGear** Recently in 2019, a further improvement for the offline phase was introduced, called TopGear [3]. It is implemented in SCALE-MAMBA since version 1.3[1], which was released in January 2019. It further improved the HighGear protocol of the Overdrive implementation, by targeting specifically the ZKPs and increasing the security guarantees provided while maintaining roughly the same performance.

**Sacrificing of triples** Even though the triples are generated and each party holds the respective shares, there is still an allowed error $e$ for each triple $(\langle x_1\rangle, \langle y_1\rangle, \langle z_1\rangle)$ so that $z_1 = x_1 * y_1 + e$. To guarantee that the triples are really sound, the correctness has to be checked. This is done with a second triple $(\langle x_2\rangle, \langle y_2\rangle, \langle z_2\rangle)$ in a procedure called sacrificing, which uses the second triple to guarantee the soundness of the first. In SPDZ that is done as follows:

1. Choose a random variable $r$ and open it to the other parties. This is done by each party generating a random share $r_i$ and then calculating $r = r_1 + ... + r_n$.
2. The parties calculate $p = r * \langle x_1\rangle - \langle x_2\rangle$ and $\sigma = \langle y_1\rangle - \langle y_2\rangle$ and partially open the results afterwards
3. The parties calculate $\phi = r * \langle z_1\rangle \ - \langle z_2\rangle - \sigma * \langle x_2\rangle - p * \langle y_2\rangle - \sigma * p$ and then partially open the result
4. If $\phi$ is 0 then the triples are correct and it is proceeded with $(\langle x_1\rangle, \langle y_1\rangle, \langle z_1\rangle)$ as a correct, secretly shared and not revealed triple. Otherwise the system detected a malicious adversary and it aborts.

Remark: As shown later in chapter 3.6, addition of two secretly shared values and multiplication of a secretly shared value with a public constant can be done individually by every party, so that, when later revealing the results of each party, the correct result can be computed by the parties together. Accordingly, beforehand operations can be assumed as working as seen later.

**Correctness**: Simplifying $\phi$, it can be see that every term disappears (given that the triples are proper triples):

$$\phi = t{*}z_1{-}z_2{-}y_1{+}y_1{*}x_2{+}y_2{*}x_2{-}t{*}x_1{*}y_2{+}x_2{*}y_2{-}t{*}x_1{*}y_1{+}t{*}x_1{*}y_2{+}x_2{*}y_1{-}x_2{*}y_2$$

$$= t*(z_1-x_1*y_2+x_1*y_2-x_1*y_1)+y_1*x_2-x_2*y_1+x_2*y_2-z_2+x_2*y_2-x_2*y_2$$

$$= t*(z_1 - x_1*y_1) + x_2*y_2 - z_2$$

Therefore, $\phi = 0 \Leftrightarrow x_1 * y_1 = z_1 \wedge x_2 * y_2 = z_2$, so this only holds if both triples are correct.

In this process the triple $(\langle x_1\rangle, \langle y_1\rangle, \langle z_1\rangle)$ wasn't revealed, so now it can be used for the multiplication knowing that one the one hand it is still secretly shared and unknown and on the other hand that it was not modified by an adversary. **Sacrificing of triples in SCALE-MAMBA**

In later versions of SPDZ, since [18], and in the current implementations of SCALE-MAMBA, a slightly modified version of aforementioned protocol is used. The idea is to not sacrifice a random second triple, to guarantee the correctness of the first one, but to sacrifice a similar triple. For this, the triples are directly generated in pairs, so that there are are two triples $(\langle x\rangle, \langle y\rangle, \langle z\rangle)$ and $(\langle x'\rangle, \langle y\rangle, \langle z'\rangle)$ with $z = x * y$ and $z' = x' * y$. With $y_1 = y_2$, the formula from SPDZ becomes directly much simpler.

1. Choose a random variable $r$ and open it to the other parties. This is done by each party generating a random share $r_i$ and then calculating $r = r_1 + ... + r_n$.
2. The parties calculate $p = r*\langle x\rangle - \langle x'\rangle$ and partially open the result afterwards
3. The parties calculate $\phi = r * \langle z\rangle \ - \langle z'\rangle - p * \langle y\rangle$ and then partially open the result
4. If $\phi$ is 0 then the triples are correct and it is proceeded with $(\langle x\rangle, \langle y\rangle, \langle z\rangle)$ as a correct, secretly shared and not revealed triple. Otherwise the system detected a malicious adversary and it aborts.

**Correctness**: Parallel to before, simplifying $\phi$ leads to the same implications:

$$\phi = r*z - z' - p*y = r*z - z' - (r*x - x')*y = r*(z - x*y) + (x'*y - z')$$

Therefore, $\phi = 0 \Leftrightarrow x * y = z \wedge x' * y = z'$

**Comparison** This small improvement in the protocol of sacrificing nearly halved the mathematical operations necessary for the scarification process. This is huge, because the triple generation is one of the most time consuming task, so it sped up the offline phase. In following table, the differences in needed operations between both sacrificing protocols can be seen:

| | SPDZ | SCALE-MAMBA |
|---|---|---|
| Additions | 6 | 3 |
| Multiplications with constants | 5 | 3 |
| Partial reveals | 3 | 2 |

Alone the changes in the usage of the zero knowledge proofs already improved the run time of the offline phase, in comparison between SPDZ and SCALE-MAMBA, by a factor of 1000. Together with the changes in the the sacrificing and with a lot more smaller changes that will remain undocumented in this paper, the run time of the offline phase improved by a huge margin in the last years.

### 3.6 Online Phase

**Preliminaries** After the offline phase is executed successfully, the online phase takes place, which contains the evaluation of the given function with the provided input secrets. It is important to note, that the online phase is nearly perfectly optimized given an earlier execution of the offline phase. The online phase has a linear run time complexity depending on the amount of operations needed for the evaluation of the function, and therefore there is not a really big performance impact. It is also noteworthy, that in the online phase only computational and no information theoretical problems are solved. All this leads to the fact, that there is not really any change in the online phase in the SPDZ like protocols in the last years, therefore the online phase is the same like in the earlier version of SPDZ [10].

We assume following preliminaries are given after the execution of the offline phase:

1. There are $n$ parties $P_1, ..., P_n$
2. There is a given function $f(a_1, ..., a_m)$ which the parties want to evaluate together. The inputs are secret, and each input is hold by exactly one party.
3. There is a global MAC key $\alpha$ that is secret to every party and additively shared, so that every party $P_i$ has a share $\alpha_i$ so that $\alpha = \alpha_1 + ... + \alpha_n$
4. There are $m$ secret inputs $a_1, ..., a_m$ which are each hold by exactly one party
5. There is a sufficiently large queue of beaver triples which are secretly shared

First, the secret inputs $a_1, ..., a_m$ are additvely shared with the other parties as described before in chapter 3.3. So now, each party $P_j$ holds a share $(a_i)_j$ of every secret $a_i$.

Following, it is going to be demonstrated, how addition and multiplication are done in this environment. These two operations are sufficient, because a group with addition and multiplication is already touring complete, i.e. it is possible to express every function with a combination of additions and multiplications. Furthermore, addition and multiplications with constants are also discussed, because they can be done in a optimized way that saves computation time.

**Addition** Assumed there are two values $a$ and $b$ which should be added together, and both are secret inputs from two different parties. Naturally, both of these values are additively shared between all the parties, i.e. every party has a share $\langle a_i \rangle$ and a share $\langle b_i \rangle$.

Then every party $P_i$ adds up their local shares, so that:

$$\langle c_i \rangle = \langle a_i \rangle + \langle b_i \rangle = ((\delta_a + \delta_b), (a_i + b_i), \gamma(a)_i + \gamma(b)_i))$$

**Correctness:** Afterwards, $c = a + b$ is additively shared between the parties and can be used for further computations, because when fully revealing $\langle c \rangle$ following statements hold:

1. $a + b = (a_1 + ... + a_n) + (b_1 + ... + b_n) = (a_1 + b + 1) + ... + (a_n + b_n) = c$
2. $\delta_a + \delta_b = \delta_c$
3. $\gamma(a) + \gamma(b) = \alpha * (\delta_a + a) + \alpha * (\delta_b + b) = \alpha * (\delta_a + \delta_b + a + b) = \alpha * (\delta_c + c) = \gamma(c)$

**Runtime:** The runtime is constant, because every party has to do 3 additions per execution of the protocol.

**Addition with constant** Adding a public constant $c$ to a secretly shared value $\langle a \rangle$ is really simple in this representation, because $\gamma(a) = \alpha(a + \delta)$ holds:

$$\langle a \rangle + c = (\delta - c, (a_1 + c, a_2, ..., a_n), , (\gamma(a)_1, ..., \gamma(a)_n)$$

**Correctness:** It can be seen, that the constant is only added to one share $a_1$ of the value and subtracted from $\delta_a$. $\langle a + c \rangle$ is properly secretly shared because:

1. $a + c = (a_1 + ... + a_n) + c = (a + 1 + c) + a_2 + ... + a_n$
2. $\gamma(a) = \alpha * (a + \delta_a) = \alpha * ((a + c) + (\delta_a - c)) = \gamma(a + c)$

**Runtime:** The runtime is constant, because every party has to do 1 or 2 additions per execution of the protocol.

**Multiplication with constant** Multiplication with a public constant is as well really simple, and can also be done by each party individually without any communication between the parties. It is defined as:

$$c * \langle a \rangle = \langle c * a \rangle = (c * \delta, (c * a_1, ..., c * a_n), (c * \gamma(a)_1, ..., c * \gamma(a)_n)$$

**Correctness**: We have to examine the different parts, to see each parts correctness:

1. $c * a = c * (a_1 + .... + a_n) = c * a_1 + ... + c * a_n$
2. $c * \gamma(a)_1 + ... + c * \gamma(a)_n = c * \gamma(a) = c * (\alpha * (a + \delta_a)) = \alpha * (c * a + c * \delta_a)) = \gamma(c * a)$

**Runtime:** The runtime is constant, because every party has to do 3 multiplications per execution of the protocol.

**Multiplication** Multiplication is the only operation, except for distributing the shares and calculating and verifying the end result, where the parties have to interact in the online phase.

Assumed there are two secretly shared values $\langle a \rangle$ and $\langle b \rangle$ which should be multiplied with each other. Naturally, each party $P_i$ holds the shares $\langle a_i \rangle$ and $\langle b_i \rangle$.

For each multiplication operation, one of the secretly shared Beaver triples $(\langle x \rangle, \langle y \rangle, \langle z \rangle)$ generated in the offline phase is needed, which is taken out of the queue.

The multiplication is done with following protocol:

1. The parties calculate $\epsilon = \langle a \rangle - \langle x \rangle$ and partially open it
2. The parties calculate $\delta = \langle b \rangle - \langle y \rangle$ and partially open it
3. The parties calculate $\langle c \rangle = \langle z \rangle + \epsilon * \langle b \rangle + \delta * \langle a \rangle + \epsilon * \delta$

**Correctness:**

$$\langle a \rangle * \langle b \rangle = \langle (\langle a \rangle - \langle x \rangle + \langle x \rangle) \rangle * \langle (\langle b \rangle - \langle y \rangle + \langle y \rangle) \rangle$$

$$= (\epsilon + \langle x \rangle) * (\delta + \langle y \rangle)$$

$$= \langle x \rangle * \langle y \rangle + \epsilon * \langle b \rangle + \delta * \langle a \rangle + \epsilon * \delta = \langle c \rangle$$

As it can be seen, the computations are correct and calculate the expected result, and at the same time no information about $\langle a \rangle$ or $\langle b \rangle$ is revealed.

**Runtime:** With the help of the beaver triple, it was possible to express a multiplication as a combination of 5 additions, 2 multiplications with constants and 1 multiplication of two constants by each party. Additionally two values were partially opened, but this can be done in constant time too. So a multiplication can be done in $O(1)$.

**Determining the Result and Verifying Correctness** At the end of the circuit evaluation, the total result $r$ has to be computed. After all computations were executed, every party has a share of the final result $r_i$, a share of the tag $\gamma(r)_i$) of the final result and a share of the global MAC key $\alpha_i$. The verification is done as follows:

1. Every party opens their share $r_i$ and their share of the MAC value $\gamma(r)_i$)
2. Every party computes $r = \sum_{i=1}^{n} r_i$
3. Every party computes $\gamma(r)_i = \sum_{i=1}^{n} \gamma(r)_i$
4. Every party computes $\sigma_i = \gamma(r)_i - \alpha_i * r$ and shares it to all the other parties
5. Every party computes $\sum_{i=1}^{n} \sigma_i$ and checks if it is 0. If it is 0, the computation was correct.

**Correctness:**

$$\sum_{i=1}^{n} \sigma_i = \sum_{i=1}^{n} \gamma(r)_i - \alpha_i * r$$

$$= \sum_{i=1}^{n} \gamma(r)_i - \sum_{i=1}^{n} \alpha_i * r$$

If the calculations were correct, the MAC was shared properly and not modified, and the tag values were not modified, then the difference between both sums will be 0. If it is not 0, the protocol detects that there is a malicious adversary and aborts.

**Runtime Complexity** Because all the mathematical operations can be executed in O(1) given the preliminaries from the offline phase, the mathematical evaluation of the circuit can be executed in O(d) with d being the amount of operations the circuit consists of. Additionally, each party can share a secret in O(n), so the total complexity class of the online phase is linear. Therefore, the online phase is really well scalable.

### 3.7 Problems and upcoming changes

**Upcoming Changes** SCALE-MAMBA is a research system, so there will be a lot more research spend on improving the system further. An important aspect is the usability in the practice. Because online and offline phase are integrated into each other, the complete runtime sometimes takes a lot of time. This is especially the case in full threshold access structures. Because of the integration, it is right now not possible to execute the offline phase earlier. One of the currently ongoing improvements on SCALE-MAMBA tackles exactly this problem. The developer team tries to give MAMBA the possibility to compile code just in time as mentioned in [1]. That would offer the possibility, that the offline data is calculated before and then, when the need arises, a computation could be specified in MAMBA and in-time compiled and executed, without having to wait for the expensive offline calculations.

**Problems** It was shown recently in [21] that there are security flaws in SPDZ 2.0 and that its secret sharing is predictable. It doesn't seem to be the case that these imperfections are already fixed in SCALE-MAMBA.

Additionally, a single cheater can break the whole system because it is abort-secure. in [24] an extended protocol is suggested to detect and expel cheaters if one is detected and afterwards proceed with the protocol. This would be an important feature for a real-world use case, so the protocol doesn't get stuck in a loop of abortion.

## 4 Implementations

Because SCALE-MAMBA was only released in 2018, there are not a lot documented use cases in praxis. Nevertheless, because of the structural and algorithmic similarities to the earlier versions of SPDZ, taking a look on SPDZ implementations should give a general understanding of the possible usage of SCALE-MAMBA.

### 4.1 SPDZ 2.0 in medical healthcare

In medical healthcare, preserving privacy plays a huge role because patient records hold highly sensitive data, thus, its a natural area for the application of privacy protecting technologies. In [2] a clinical decision support system was implemented with SPDZ 2.0. The idea is that the system provides the optimal treatment for a patient with HIV based on the records of similar patients in different hospitals. To preserve privacy, the hospitals compute the efficiency of one treatment for one patient together with SMPC.

The researchers used a setup of 20.000 patient records split between the hospitals, and were able to compute the effectiveness of 100 treatments for one patient in 24 minutes in the online phase. For calculating the effectiveness of one treatment, their algorithm needed 40 million multiplications. They only implemented the online phase of SPDZ, but showed that it scaled linearly with the amount of patient records. In the paper the amount of parties participating in the computation was not specified, but because the speed of the computation appears to be quite good, it can be assumed that a really simple setup is used that does not create a lot of overhead.

Supposedly, the offline phase took them around 22 minutes.

### 4.2 Other implementations

In [12] an implementation of biometric-based authentication, i.e. iris and face authentication, with SPDZ was shown. The researchers were able to improve existing solutions with the use of SPDZ.

[7] implemented typical machine learning algorithms like linear and logistic regression with SPDZ and showed that it outperforms a competing protocol Obliv-C [27] and matches SecureML [20]. So it can be assumed, that an implementation with SCALE-MAMBA would show even better results.

## 5 Comparison

In the sector of multi party protocols with highest possible security, i.e. active and full-threshold security, SCALE-MAMBA, and before SPDZ, seem to be the universally accepted state of the art protocols. As already seen before, SCALE-MAMBA has several improvements in run-time and security compared to its predecessors.

Nevertheless, the requirements are not always the same, so depending on the use-case the choice of protocol might be depend. No complete comparison to other protocols will be, but this section should transmit the parameters which should be considered when making this choice.

**Two party protocols** There are a lot of protocols, which are restricted to a two party setup. As shown in [21], frequently used protocols like TinyLEGO [14] and DUPLO [19], which are both protocols based on garbled circuits, and ABY [11], which combines secret sharing and garbled circuits, all outperform SPDZ 2.0 in terms of information loss regarding compression and entropy metrics.

**Passive Security** There are protocols, for instance SCAPI [13], which only implement passive security, i.e. they are not secure versus malicious adversaries. In a setup with looser security requirement a different protocol might be better suited because a lot of verification can be omitted.

**Different Access structures** Full-threshold security is not always a desired property. While SCALE-MAMBA can be configured for different access structure, different protocols optimized for these access structures might outperform it.

## 6 Resume

### 6.1 Outlook

There are a lot of future applications for Secure Multiparty Computation. With the theoretical research of the early years turning to more praxis-oriented research, today's SMPC protocols have a bright future. With already more and more applications starting to use SMPC in systems that need to guarantee security of private data, and with the protocols further improving in performance and security, the list will only continue growing.

SCALE-MAMBA is one of the first systems that can is functional and performant enough to be used in real-world scenarios, and, thus, it will probably not take a lot more years until there will be proper commercial solutions. And if big companies start to adapt and use these theoretical foundations, then, in my opinion, SMPC will be a really important tool in computer science and especially in distributed environments. There are a lot more possible applications that could be coming in the next years, for example in the financial sector or in inter-governmental collaborations.

SMPC can be a technology that shapes the future, or it can be a stepping stone for new technologies to emerge. But definitely, privacy and security already play a huge role in our society, and this will not diminish in any close future.

### 6.2 Conclusion

In this paper we could see how SMPC works in general, and, especially, how SCALE-MAMBA works and how it evolved over the years in comparison to its predecessor, SPDZ. It was shown, how the protocol is split into a primary offline phase in which the computational expensive calculations are executed and all the preparations are done for the later circuit evaluation, like calculating the Beaver Triples, and how afterwards in the online phase the circuit is evaluated, how the triples are used to execute a multiplication without sharing the secret inputs and how the results can be verified at the end. Nevertheless, SCALE-MAMBA is optimized for a setup with active and full threshold security, so if the use case varies, there might be a more suitable SMPC protocol.

## References

1. Aly, A., Keller, M., Orsini, E., Rotaru, D., Scholl, P., Smart, N., Wood, T.: SCALE-MAMBA Documentation v1.3. https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf (2019)
2. Attema, T., Mancini, E., Spini, G., Abspoel, M., de Gier, J., Fehr, S., Veugen, T., van Heesch, M., Worm, D., De Luca, A., Cramer, R., Sloot, P.: A new approach to privacy-preserving clinical decision support systems for hiv treatment (10 2018)
3. Baum, C., Cozzo, D., Smart, N.P.: Using topgear in overdrive: A more efficient zkpok for spdz. Cryptology ePrint Archive, Report 2019/035 (2019), https://eprint.iacr.org/2019/035
4. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91. pp. 420–432. Springer Berlin Heidelberg, Berlin, Heidelberg (1992)
5. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. Cryptology ePrint Archive, Report 2010/514 (2010), https://eprint.iacr.org/2010/514
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277 (2011), https://eprint.iacr.org/2011/277
7. Chen, V., Pastro, V., Raykova, M.: Secure Computation for Machine Learning With SPDZ. arXiv e-prints arXiv:1901.00329 (Jan 2019)
8. Cramer, R., Damgård, I., Xing, C., Yuan, C.: Amortized complexity of zero-knowledge proofs revisited: Achieving linear soundness slack. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 479–500. Springer International Publishing, Cham (2017)
9. Cramer, R., Damgård, I.B., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing, p. 51–103. Cambridge University Press (2015)
10. Damgard, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535 (2011), https://eprint.iacr.org/2011/535
11. Demmler, D., Schneider, T., Zohner, M.: Aby - a framework for efficient mixed-protocol secure two-party computation (01 2015)
12. Droandi, G., Barni, M., Lazzeretti, R., Pignata, T.: Semba:secure multi-biometric authentication (03 2018)
13. Ejgenberg, Y., Farbstein, M., Levy, M., Lindell, Y.: Scapi: The secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629 (2012), https://eprint.iacr.org/2012/629
14. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B., Trifiletti, R.: Tinylego: An interactive garbling scheme for maliciously secure two-party computation. Cryptology ePrint Archive, Report 2015/309 (2015), https://eprint.iacr.org/2015/309
15. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing. pp. 291–304. STOC '85, ACM, New York, NY, USA (1985), http://doi.acm.org/10.1145/22145.22178

16. Goldwasser, S., Micali, S.: Probabilistic encryption and how to play mental poker keeping secret all partial information. pp. 365–377 (01 1982)
17. Keller, M., Orsini, E., Scholl, P.: Mascot: Faster malicious arithmetic secure computation with oblivious transfer. Cryptology ePrint Archive, Report 2016/505 (2016), https://eprint.iacr.org/2016/505
18. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making spdz great again. Cryptology ePrint Archive, Report 2017/1230 (2017), https://eprint.iacr.org/2017/1230
19. Kolesnikov, V., Nielsen, J.B., Rosulek, M., Trieu, N., Trifiletti, R.: Duplo: Unifying cut-and-choose for garbled circuits. Cryptology ePrint Archive, Report 2017/344 (2017), https://eprint.iacr.org/2017/344
20. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. pp. 19–38 (05 2017)
21. Resende, J., Sousa, P., Martins, R., Antunes, L.: Breaking mpc implementations through compression. International Journal of Information Security (01 2019)
22. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 239–252. Springer New York, New York, NY (1990)
23. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (Nov 1979), http://doi.acm.org/10.1145/359168.359176
24. Spini, G., Fehr, S.: Cheater detection in spdz multiparty computation. vol. 10015, pp. 151–176 (08 2016)
25. Yao, A.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167 (Oct 1986)
26. Yao, A.C.C.: Protocols for secure computations. 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982) pp. 160–164 (1982)
27. Zahur, S., Evans, D.: Obliv-c: A language for extensible data-oblivious computation. Cryptology ePrint Archive, Report 2015/1153 (2015), https://eprint.iacr.org/2015/1153