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. To do that, it first introduces SMPC, afterwards SPDZ, the predecessor of SCALE-MAMBA, and, after taking a detailed look at SCALE-MAMBA, compares them. It focuses on the algorithmic foundations of SCALE-MAMBA, 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 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[TODO: LINK] this is done by specific people counting a small amount of votes and 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. It is a field of research that emerged in 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, a system developed mainly by the Katholieke Universiteit Leuven. This system is one of the forerunners in SMPC 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 Cryptographic Basics

Before delving deeper into SMPC, this section will shortly recap on cryptography basics that might be unknown to the reader. These basics are not solely used by SMPC, but nevertheless, play a huge role in a lot of protocols.

2.1 Message Authentification Codes

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 $(\mathrm{MAC}, \mathrm{VER})$:

- The MAC algorithm $\mathrm{MAC}(K, m) \to t$ takes a key K and a message m and outputs a tag t
- The verficiation algorithm $\mathrm{VER}(K, m, t) \to 0/1$ takes a key K, a message m and a tag t and outputs accept(1) or reject(0)

The correctness can be easily seen: $\forall K : \mathrm{VER}(K, m, \mathrm{MAC}(K, m)) = 1$

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

$$\mathrm{MAC}(\alpha, m) \to \alpha * m$$

$$\mathrm{VER}(\alpha, m, t) \rightarrow (1 \Leftrightarrow t = \alpha * m)$$

This algorithm has the disadvantage of high storage complexity, but therefore it doesn't take a long time to calculate and it is symmetric so the same key can be used for creating the tag and verifying it. The symmetry is used for an advantage later on in the protocol.

2.2 Homomorphic Encryption

Encryption as a basic concept is simple. Similiar to MACs, a tupel (E,D) of algorithms is needed.

- The enncryption algorithm $\mathrm{E}(K, m) \rightarrow c$ takes a key K and a message m and outputs a cyphertext t that is the encryption of the message m
- The decryption algorithm $\mathrm{D}(K, c) \rightarrow m$ takes a key K and a cyphertext m and outputs the decrypted message m, assuming c was not modified

Alternatively, in a public key setting, the key for encryption is public and there is a different key for decryption that is privat.
The problem is, that modifying the cypertext in any way makes the decryption useless. In some cases it is desirable do do computations on cypertexts, because then it is possible to make calculations without revealing the unencrypted data. That is why Homomorphic Encryption was developed. Let's assume that we have a ring $(R, +, *)$ as our message space. Fully Homomorphic Encryption (FHE) is preserving the ring structure. Therefore, it is possible to do additions and multiplications of cypertexts and when decrypting the result it is the same value as when doing the same additions and multiplications on the real values.
Althought FHE has amazing properties for a lot of usecases, in praxis it is way too slow. Because of all the overhead generated by the encryption scheme to guarantee the preservation of the ring structure, the current algorithms are not feasible for most applications.

2.3 Somewhat Homomorphic Encryption

That is where Somewhat Homomorphic Encryption (SHE) becomes relevant. SHE only supports one operation, and in most cases that is addition. This takes away a lot of possibilities but, therefore, the performance gain is immense. So immense, that SHE is used a lot in practice, for example in SCALE-MAMBA.
The SHE scheme used in SCALE-MAMBA, is based on the so called Brakerski-Gentry-Vaikuntanathan cryptosystem(BGV).

3 SMPC Basics and History

3.1 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(how can two players play poker while not being in the same physical location and making sure no one is cheating) and Yao's millionaire problem(how can can two millionaires decide who is richer than the other while not revealing their respective wealth).
These explorations lead to a group of basic protocols which established the area of Secure Multi-Party Computation(SMPC).

3.2 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.

3.3 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 [**?**]. The basic idea is that, in a two party setting, one party, Alice, garbles 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.

3.4 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 [**?**].
The basic idea of this approach is that multiple parties split their secrets into so called shares and then distribute them to the other parties. In Shamir's scheme this is done by generating random polynomials with a degree depending on the security level and then calculating the shares as points in this polynomial. 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.
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.

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.

3.5 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 something because of his own interest. In SMPC it is expected that the adversary is not only participating as one party, but could 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 to 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.

3.6 Malicious adversary

While this adversary also wants to gain all the information, he is not obeying to 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 decide between these two types of adversary, because it shows the different securities SMPC has to guarantee. It has to guarantee that no information is unwillingly shared with an illegal party and at the same time verify that the final result has a correct value. Most SMPC protocols try to guarantee safety versus both type of adveraries, bvut there are also approaches that only use passive security, i.e. only protecting versus non malicious adversaries.

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 measurements that have to be taken and, thus, for runtime 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. Here, n is the number of total parties, and t is 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 it 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 really important, because it has a realy big impact on performance and on the desired security level.

4 SCALE-MAMBA

Overview In the last years, research in SMPC is constantly ongoing, and there are new approaches or improvements to old approaches appearing regulary. But

most of these publications only focus on a small area of the whole picture. SCALE-MAMBA[**?**] is a project that tries to combine the state-of-the-art research into one system. That means 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 an overview how the process works in general, which are the main, especially which are newest and best, components and where are the problems and bottlenecks.
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 the parties to prepare for the evaluation. This includes already calculating values for possible multiplications and MACs, so that the online phase can be executed faster. In this phase the expensive calculations are done, like public key cryptography or authentifications, so that the online phase only needs basic primitives. This phase is not completly offline, because the parties still exchange some information which are later needed for the evaluation. But this phasae 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 completly integrated. Therefore, it is not possible anymore to execute them separately like in SPDZ for example. But it is a much closer approach to a real world application.

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 except one party could be corrupted, and still no information is leaked, because the evaluation is aborted.

MAMBA For this paper the more interesting part is the algorithmic part. But still, the importance of the compiler shouldn't be undervalued. Because while the algorthmic foundations might be more important from a theoretical point of view, the compiler, written in C++, enables the user to really use these technologies in a system where he can rely on the security gurantees given by the algorithms. The programs can be written for MAMBA in a syntax similiar 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 is the compiler which then translates the instruction to bytecode and executes them. MAMBA is the part that gives the system the possibility to be properly used and tested. It makes it easy, to change the parameters, to be able to execute comparable tests, and to have a abstraction level for the user.

Secret sharing As mentioned before, SCALE-MAMBA, and in general similiar protocols like SPDZ, use linear secret sharing.
Given party P_i which holds a secret value $a \in \mathcal{F}_p$ and a share of the global MAC key α. 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 to each party P_j δ_a and the shares a_j and, $\gamma(a)_j$
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, so 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.

Revealing secrets For obvious reasons, the secrets also have to be revealed again. SCALE-MAMBA uses two different types of revealing them.

Partial Reveal All players send their respective shares of a value, but not the tag share, to one chosen party P_i which summs 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.

Offline phase There are two main things done in the offline phase. The first one is the generation of a global MAC key α. This MAC key is later used, to

authenticate the different shares at the end of the computation. This key is also distributively generated, i.e. only if all parties pool together their shares they can recover its value.

The second and and most important part in the offline phase, 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 the online phase to guarantee that multiplications can be done without revealing the secret inputs of the different parties. For each multiplication, one of the triples is needed.

Beaver Triples Beaver triples 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 the online phase 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:

GENERATION OF BEAVER TRIPLES: Include protocol on how triples are distributively generated without revealing them

Zero Proofs of Knowledge 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 Proofs of Konwledge Protocol(ZPoKP).

In the offline phase, ZPoKP are used to guarantee that the Beaver triples are calculated correctly before the sacrificing, or atleast that they dont divert more than a specified error e from the expected result.

ZPoKP are a technique that was introduced in 1985 [**?**]. 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(in this scenario 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 $a + e * yandf(z)$ 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 firt step she chose a random element s and calulated a=f(s). This information is independent on her secret x, so she didn;t share any information. When calculating z=s+e*x, she also didnt share any information regarding x, as long it is guaranteed that Bob didnt have any knowledge about the s. So under the assumption that Alice chooses a new random s each time the protcol is executed, she doesn't share any information regarding her secret x.

It is not really important for this thesis, which ZPoKP is exactly used, but it should be said that the protocol used in SCALE-MAMBA is based on Schnorr's protocol.

First improvement with MASCOT In one of there earlier works, BDOZ, pairwise MACs were used to authentificate the secret sharing between the parties. It used a pairwise multiplication protocol with linear homomorpic encryption. To guarantee the data integritym pairwise zero-knowledge proofs were used, in total O(n2) of them. In SPDZ instead of HE, SHE based on the BGV ecnryption scheme was used, reducing the amount of ZK proofs per triple by a factor of n and as well the overhead per triple, because they used a single ZKPoK for proving thousand of triples. Because the offline phase was the weak point of the existing SPDZ implementations, in regards to runtime while still managing a sufficiently secure system, in 2016 MASCOT [**?**] 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.

Overdrive Shortly afterwards in 2017 a new protocol for the offline phase called Overdrive was introduced [**?**], which once again got better results with SHE instead of OT, improving the run time at least by a factor of 6. Basically, it is a really similiar protocol to MASCOT, but just uses SHE in the places where MASCOT used OT. So explaining Overdrive already should give a good

impression how MASCOT works.
Overdrive includes two parts, a protocol called Low Gear protocol for small numbers, and a proctocol 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 the ZKPoK usage in Overdrive and in SPDZ is, that the parties not anymore prove multiple smaller statements between each other with ZKPoK, but instead make a joint statement and prove thi one toegether. Because ZKPoK are really expensive, this makes a big difference in the overall performance.
This way, Overdrive did not improve the amount of communication channels used, but it decreased the computational costs by a factor n.
A second big strategy used by SPDZ and Overdrive is the so called Soundness Slack. Because in practice it is to expensive to prove that a result is exact, so it is sufficient to prove that the value, here the product of the beaver triple, exact but with the legalization of a small error. Later on, this error, if existing and introduced by an adversary, will be detected by a different technique called sascrificing.
Overdrive uses as well drowning, which is the technique of adding noise to a secret sharing. Because otherwise, the noise of the SHE function would reveal information over the secret. So the noise is drowned by introducing an additional random noise that is bigger than the maximum possible noise.

TopGear Recently, an even newer improvement for the offline phase was introduced, called TopGear [**?**]. It is implemented in SCALE-MAMBA since version 1.3[**?**], which was released in January 2019. It further improved the HighGear protocol of the Overdrive implementation, by targeting specifically the ZKPoKs.

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 ϕ 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 [X], 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 ϕ, 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 [**?**], 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 ϕ 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 ϕ 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 of Sacrificing This small improvement in the protocol 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 between both sacrificing protocols can be seen:

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

In SPDZ: Sacrifice in online phase???

Online Phase

Preliminaries Now it is assumed that the offline phase was succesfully executed, so all preliminaries are given to execute the online phase. It's main goal is to evaluate a specified circuit with secret inputs by the different parties. So this is the heart of the system, the part where the real computatiuon is executed.
It is important to note, that the online phase is nearly perfectly optimized when assuming the offline phase was carried out succesfully. The online phase has a runtime complexity in O(n), 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 [**?**].

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 α that is secret to every party and additively shared, so that every party P_i has a share α_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 [Link to secret sharing in SM XXX]. 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, so there is no need for any other operation[TODO: ADD reference]. Furthermore, addition and multiplications with constants are also discussed, because they can be done in a simple 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. $\gamma(a)+\gamma(b) = \alpha*(\delta_a+a)+\alpha*(\delta_b+b) = \alpha*(\delta_a+\delta_b+a+b) = \alpha*(\delta_c+c) = \gamma(c)$

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 δ_a. $\langle a + c \rangle$ is properly secretly shared because:

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

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. $c*\gamma(a)_1+...+c*\gamma(a)_n = c*\gamma(a) = c*(\alpha*(a+\delta_a)) = \alpha*(c*a+c*\delta_a)) = \gamma(c*a)$

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 has the shares $\langle a_i \rangle$ and $\langle b_i \rangle$.
For each multiplication operation, 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 completly unrelated to each other, and the sacrificing 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 $p = t * \langle x_1 \rangle - \langle x_2 \rangle$ and $\sigma = \langle y_1 \rangle - \langle y_2 \rangle$ and partially open the results afterwards
3. The parties calculate $\phi = t * \langle z_1 \rangle \ - \langle z_2 \rangle - \sigma * \langle x_2 \rangle - p * \langle y_2 \rangle - \sigma * p)$ and then partially open the result
4. If ϕ is 0 then the triples are ok and it is proceeded with $(\langle x_1 \rangle, \langle y_1 \rangle, \langle z_1 \rangle)$. Otherwise the system detected a malicious adversary and it aborts.

Correctness: If we simplify ϕ, we can see that every term disappears (given that the triples are proper triples):

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

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

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

So we can see that $\phi = 0 \Leftrightarrow x_1 * y_1 = z_1 \wedge x_2 * y_2 = z_2$, so we know the triple is correct.
In this process we didn't reveal the triple $(\langle x_1 \rangle, \langle y_1 \rangle, \langle z_1 \rangle)$, so now we can use it 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.

There is a small improvement in SCALE-MAMBA compared to SPDZ regarding the triple generation that was added later in [**?**]. The idea is to not sacrifice a random second triple, to guarantee the correctness of the first one, but to sacrifice a similiar triple. For this, the triples in the online phase are already generated in pairs, so that there are are two triples $(\langle x \rangle, \langle y \rangle, \langle x * y \rangle)$ and $(\langle x' \rangle, \langle y \rangle, \langle x' * y \rangle)$. This way the triples can be calculated faster in the offline phase, because one half of the multiplication is the same, and still the triples keep the property to be sacrificed for each other to check the correctness of the other triple.

In SCALE-MAMBA, this triple sacrifice is also already executed in the offline phase. But for consistency the explanation remains here, to better be able to see the difference between SPDZ and SCALE-MAMBA.

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:

$$\begin{aligned} \langle a \rangle * \langle b \rangle &= \langle (\langle a \rangle - \langle x \rangle + \langle x \rangle) \rangle * \langle (\langle b \rangle - \langle y \rangle + \langle y \rangle) \rangle \\ &= (\epsilon + \langle x \rangle) * (\delta + \langle y \rangle) \\ &= \langle x \rangle * \langle y \rangle + \epsilon * \langle b \rangle + \delta * \langle a \rangle + \epsilon * \delta = \langle c \rangle \end{aligned}$$

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 α_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.

Upcoming changes to SCALE-MAMBA SCALE-MAMBA is a research system, so there will be a lot more research time 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 [TODO: link to documentation]. 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 then in-time compiled and executed, without having to wait for the expensive offline calculations.

5 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.

5.1 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 [TODO: REFERENCE] 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 computation is quite good, it can be assumed that a really simple setup is used that doesn't create a lot of overhead.

TODO: compare, what is the speed and how fast are different ones? TODO: CHECK HOW MANY HOSPITALS

Supposedly, the offline phase took them around 22 minutes.

https://www.researchgate.net/publication/328040632A_New_Approach_to_Privacy-Preserving_Clinical_Decision_Support_Systems_for_HIV_Treatment https://arxiv.org/pdf/1901.00329.p https://www.researchgate.net/publication/324077995_SEMBASEcure_multi-biometric_authentication

https://www.moodle.ch/lms/pluginfile.php/6253/mod_book/chapter/25/mpc-book.pdf

6 Comparison

a. What else is out there? b. Different use cases – when which SMPC tool is more useful; what are the strengths of SCALE-MAMBA compared to other ones? - Security! High performant for such a secure system - In some cases not the highest security is needed; application for a normal user, not for a company, have much less security requirements but much higher requirements on performance - Active security is not always required, in non-critical systems it is often sufficient to know that noone knows the users data c. Factual comparisons - YAO/LSSS –> LSSS protocols right now are more suited for the praxis

- SCAPI -SCAPI is an open-source Java library for implementing secure two-party and multiparty computation protocols (SCAPI stands for the "Secure Computation API"). It provides a reliable, efficient, and highly flexible cryptographic infrastructure. SCAPI provides only passive security

https://scapi.readthedocs.io/en/latest/intro.html

7 Resume

Outlook Their are a lot of future applications for Secure Multiparty Computation. With the threoretical research of the early years much more turning to praxis oriented research, todays SMPC protocols, and especially the successors of SPDZ, have a bright future. With already more and more applications starting

to use SMPC in systems that need to guarantee security of private data, the list will only grow more and more with the performance of the protocols and the same time the available computation power improving.
With SCALE-MAMBA being one of the first performant systems that can be really used in a real time scenario, it will probably not take a lot of 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 distrbuted environments. There are a lot more possible applications that could be coming in the next years, for example in the financial branche SMPC could play a huge role.
SMPC can be a technology that shapes the future, or it can be just a stepstone for a new technology to emerge. But definetly, privacy and security plays already a huge role in our society, and this will not diminish in any close future.

https://www.researchgate.net/publication/309817066_Cheater_Detection_in_SPDZ_Multiparty_Co
https://www.researchgate.net/publication/330091440_Breaking_MPC_implementations_through_compr

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.

References

1. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Computing 16(03), 80–85 (May 2012)
2. Kopp, O., et al.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: Proceedings of 11th International Conference on Service-Oriented Computing (ICSOC'13). LNCS, vol. 8274, pp. 700–704. Springer Berlin Heidelberg (2013)
3. Scharrer, M.: The `mwe` Package (2017), `http://texdoc.net/mwe`