A Short Introduction To MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer

Florian Raabe, Christopher Harth-Kitzerow*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: florian.raabe@tum.de, christopher.harth-kitzerow@tum.de

***Abstract*—The MASCOT protocol allows a secure multiparty computation of arithmetic circuits over a finite field. Using oblivious transfer in an arithmetic context, it creates multiplication triples which are used to compute products of additively secret-shared values. The expensive computation to securely generate these triples is done in a preprocessing phase. After a one-time setup, the protocol is based entirely on fast, symmetric cryptography. By making use of efficient oblivious transfer extensions, the total cost for multiplications is reduced. With careful consistency checks and other techniques for privacy amplification MASCOT achieves active security. It considers a dishonest majority where any number of parties can act actively maliciously.**

***Index Terms*—multiparty computation; oblivious transfer**

1. Introduction

Secure Multiparty Computation (MPC) is a cryptographic method to jointly evaluate a function on private inputs without revealing those to the other parties. To realize this functionality, one approach is to secret-share input values between all parties. Most protocols use a linear secret-sharing scheme which allows the local addition and subtraction of shares. The parties obtain a share of the corresponding operation on the secrets. A simple example is the additive secret-sharing scheme. To secret-share a value s between n parties, a tuple $(s_1, ..., s_n)$ is sampled uniformly random so that $s_1 + ... + s_n = s$. In order to compute every function in a field on these secret shares, multiplication is needed.

The **M**alicious **A**rithmetic **S**ecure **C**omputation with **O**blivious **T**ransfer protocol by Marcel Keller, Emmanuela Orsini and Peter Scholl [1] also makes use of additive sharing. It allows for efficient and secure computation of general arithmetic circuits using almost exclusively fast, symmetric cryptography. The advantage of arithmetic circuits over Boolean circuits is that secure addition can be done locally, thus not requiring any communication.

MASCOT is the first protocol to use oblivious transfer with a dishonest majority setting to generate multiplication triples in any sufficiently large field. It works with n parties and considers a corruption of up to $n - 1$ active malicious adversaries. The adversary is considered to be static, meaning that corruption can only take place before a protocol starts.

The MASCOT protocol achieves this through simple consistency checks and privacy amplification techniques which will be introduced in the following section.

2. Preliminaries

The main task in preparing the MPC protocol is the creation of multiplication triples. These are additive secret sharings of tuples $(a, b, a \cdot b, a \cdot \Delta, b \cdot \Delta, a \cdot b \cdot \Delta)$ where a, b are random values used for the multiplication and Δ is a secret-shared random global MAC key. To generate a triple, the shares for a, b and Δ can be chosen randomly by each party. How secret sharings of the products are created will be shown in Section 3.

2.1. Information-theoretic MACs

Message authentication codes (MACs) are short tags used to confirm the authenticity of a message or its sender. For this, strong universal functions can be used. In this context, information-theoretic refers to the security aspect of the MAC. Perfect security can never be achieved since the adversary can always guess a random tag. This is why the probability to find a valid tag should be $\frac{1}{2^{|n|}}$ for n-bit fields [2]. In this protocol, a secret value x is represented by

$$[\![x]\!] = (x^{(1)}, ..., x^{(n)}, m^{(1)}, ..., m^{(n)}, \Delta^{(1)}, ..., \Delta^{(n)}).$$

Each party P_i holds a random share $x^{(i)}$, a random MAC share $m^{(i)}$ and a share of the fixed MAC key $\Delta^{(i)}$, such that the MAC relation $m = x \cdot \Delta$ holds. $[\![.]\!]$ denotes the linear authenticated secret sharing scheme. To open a value, all parties broadcast their shares to one party which adds them together and publishes the result x. All parties then check the MAC by committing and opening $m^{(i)} - x \cdot \Delta^{(i)}$. These shares then need to sum up to zero in order for the check to pass. To increase efficiency, random linear combinations of the MACs can also be checked.

2.2. Oblivious Transfer

1-out-of-2 Oblivious Transfer (OT) is a protocol between two parties. The sender transmits two messages from which only one will be received. This is decided by the receiver with a choice bit. However, it remains oblivious to the sender which message was received.

doi: 10.2313/NET-2022-11-1_06

Most existing protocols require public-key cryptography to implement this functionality. The MASCOT protocol uses the concept of OT extensions from Beaver introduced in 1996 [3]. A single oblivious transfer is used in combination with a seed as an initialization. From this point on many OTs can be generated with cheap, symmetric primitives. New correlated values can be created by using a generator function and adjusting the output of the receiver. In MASCOT, this is realized with the COPE protocol which will be explained later in Section 3.1. A consistency check in form of a sacrifice is used to make it maliciously secure.

2.3. Sacrificing technique

To ensure the correctness of a secret-shared value, a correlated shared value can be used. The share of a that should be checked is masked with the other secret-shared value $\hat{a}$ by computing $p = s \cdot a - \hat{a}$ where s is a random. The resulting share p is then opened and can be checked by all parties using their part of the global MAC key. To check a triple, another share is computed shown in 1 which has to open to zero. Note that $c = a \cdot b$ and $\hat{c}$ is the correlated value. By this, some bits of the correlated value might be leaked if the adversary input inconsistent values to one of the OTs.

$$s \cdot c^{(i)} - \hat{c}^{(i)} - b^{(i)} \cdot p \tag{1}$$

3. Preparation Phase

To obtain an actively secure product-sharing protocol, MASCOT improves the passively secure protocol of Gilboa [4]. The basic concept of this is to run OT instances between every pair of parties so that every party has a share of the products in the triple. A k-bit field element is split into bits, hence k oblivious transfers are used. Still, malicious parties can provide inconsistent inputs which will lead to potentially incorrect results when the generated triples are used. In order to prevent this, two strategies are used.

First of all, the correctness of the products in the MAC generation has to be ensured. Therefore, random linear combinations of the MACs are checked immediately after the creation as well as later when opening values. Secondly, to verify the correctness of the multiplication triples a standard sacrifice technique is used where a pair of triples is checked in order to use one securely. Furthermore, the privacy of a triple can be assured by producing several triples and taking random combinations to get a uniformly random triple. These strategies are realized by differnet subprotocols shown in Figure 1.

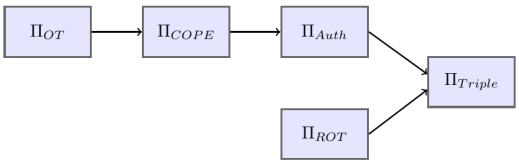

Figure 1: Dependency among subprotocols

3.1. Correlated Oblivious Product Evaluation

To obtain an additive sharing, the MASCOT protocol uses an arithmetic generalization of the passively secure OT extension of Ishai et al. [5]. The correlated oblivious product evaluation (COPE) transforms the multiplication $x \cdot \Delta$ where Δ is fixed at the start of the protocol and future iterations can create sharings for different values of x. The foundation of the COPE protocol is Gilboa's method for oblivious product evaluation.

Oblivious Product Evaluation. The concept of oblivious product evaluation (OPE) uses k sets of oblivious transfers on k-bit strings to obtain an additive sharing of the product. Let us assume P_A is the sender and P_B is the receiver. Now, P_A samples a random value t_i in each iteration and inputs the correlated value $t_i + a$ where a is the sender's input. In every OT the receiver P_B inputs one bit of their secret value. From this follows the output P_B receives in every ith iteration: $q_i = t_i + b_i * a$. Finally, both parties compute the inner product of their values $(q_i)_i$ and $(-t_i)_i$ to obtain q and t for which it holds that $q + t = x \cdot \Delta$.

COPE. The MASCOT protocol now optimizes this functionality of OPE to perform the OTs only once. Therefore, the COPE protocol is initialized at the beginning by calling the **Initialize** command. Because one party's input is still fixed, the receiver simply inputs their bits of Δ. On the other hand, the sender now does not input their secret but k pairs of random λ-bit seeds. In this case, λ is the computational security parameter and $k = \log\lfloor|\mathbb{F}|\rfloor$ the number of bits in the field.

Secondly, the protocol provides the **Extend** command which expands the original seed using a pseudo random function (PRF). This creates k bits of new, random OTs while still remaining the same receiver choice bits. To realize this, P_A uses both seeds in combination with a counter as input for the PRF to create new random seeds. Now a correlation between these outputs is created using the secret input of the sender. This masked correlation is sent to P_B who uses it to adjust their PRF output. Finally, both parties have k correlated OTs on field elements. These are then mapped into a single field element to obtain an additive sharing again.

If both parties follow the protocol, they gain an additive sharing of the multiplication. However, because the MASCOT protocol assures security against an active adversary, it has to be considered what happens if parties do not follow the protocol. Since the input of the receiving party is fixed at the start of the protocol and P_B sends no messages afterwards, there is no possibility to deviate from the protocol. Nevertheless, the sending party might use different input values in the extend phase. This is not considered a security issue because the seeds are uniformly random and the input will later be checked.

doi: 10.2313/NET-2022-11-1_06

3.2. Authentication with COPE

As shown in Figure 1, the functionality of the COPE protocol is used to create authenticated shares. Furthermore, it is used to securely open linear combinations with a MAC checking procedure. The main goal is to prevent the adversary from inputting errors in COPE and opening an authenticated share to the incorrect value. The protocol maintains a dictionary of the authenticated values and includes five commands shown in Figure 2.

Input: takes a list of values $x_1, ..., x_l$ from one party and stores them with identifiers.
LinComb: computes linear functions on values that have been input.
Open: reassembles a secret-shared value and outputs it to all parties.
Check: verifies the correctness of a value that was output by an adversary.
Abort: terminates the protocol and informs all parties, that it failed.

Figure 2: List of commands

In this functionality, the value which was opened might be incorrect. The **Check** command will confirm this. To check a MAC, the party P_i inputs an opened value y, a MAC share $m^{(i)}$ and a MAC key share $\Delta^{(i)}$. Next, they compute $\sigma = m^{(i)} - y \cdot \Delta^{(i)}$, commit and open it. If $\sigma^{(1)} + ... + \sigma^{(n)} = 0$ they continue and the opened value is correct. Otherwise, they abort.

Using the correlated oblivious product evaluation protocol to create authenticated sharings is not enough to ensure active security. To simplify, let us consider a model with only two parties P_1 and P_2. In this scenario, P_1 is honest and wants to authenticate its input x. Therefore, they initialize the COPE protocol and P_1 inputs x in the extend phase. P_2 inputs its MAC key share Δ_2. They receive t and q so that $t + q = x \cdot \Delta_2$. Following both parties define MAC shares $m_1 = x \cdot \Delta_1 + t$ and $m_2 = q$ so that clearly $m_1 + m_2 = x \cdot \Delta$. To create shares of x, P_1 simply generates random additive shares and sends one to P_2. Because the shares and the MACs are linear, both parties can compute linear combinations on authenticated values locally.

Although this is passively secure, if P_1 is actively malicious, it can choose what value to open at the time of opening a value. The party is not committed to opening a particular value and therefore it is not a secure realization of the functionality. To solve this problem, it suffices to authenticate another random value and check a random linear combination of all MACs during the input phase. This requires two changes in the Input stage. P_1 samples a random dummy input x_0 and authenticates it with the other inputs. In addition to this, after computing the MACs using the COPE protocol, P_1 opens a random linear combination of the inputs $x_0, ..., x_l$ and the MAC is checked by all parties. x_0 masks the actual inputs. Hence, P_1 cannot later open to a different value and is committed to their inputs during the Input stage.

Even though the secret values are masked, only random combinations of inputs can be checked. This could be used as an advantage because the check just relates to the randomly weighted sum of the vectors. With a probability of $\frac{1}{|\mathbb{F}|}$ there is one bit in the input vector that does not affect the MAC check. This results in two different vectors and the adversary could decide later which value to open. This can be neglected depending on the security parameter in which the subtrahend comes from the number of possible pairs where the bits are different. The protocol still securely implements the functionality with a statistical security parameter of $\log |\mathbb{F}| - 2 \log \log |\mathbb{F}|$. Note that a repeated check can ensure statistical security of $\log |\mathbb{F}|$.

Finally, the protocol can easily be extended to the use with n parties. When a party P_j inputs a value, they run COPE with every other party $P_i \neq P_j$. Naturally, they provide their MAC key share $\Delta_{(i)}$ as input. This allows P_j to obtain an authenticated share under the global MAC key $\Delta = \Delta_1 + ... + \Delta_n$. Through this, more possibilities emerge where a corrupted party might cheat and deviate from the protocol. They could for example provide inconsistent x's or use an incorrect share of Δ when authenticating other parties input. This is not problematic because except with a probability of $\frac{1}{|\mathbb{F}|}$ the MAC check will fail in the Input stage if this happens.

3.3. Generation of Multiplication Triples

The functionality described before is now being used to generate multiplication triples. In more detail, a triple $(\mathbf{a}, b, \mathbf{c})$ with $b \in \mathbb{F}$ and $\mathbf{a}, \mathbf{c} \in \mathbb{F}^\tau$ will be created. For k-bit statistical security in a k-bit field, it is sufficient to use $\tau = 4$. Note that $\tau = 3$ suffices for $\frac{k}{2}$-bit statistical security. This is due to the probability of passing the sacrifice check and the probability of distinguishing the output distribution from random. By multiplying these the number of triples that have to be combined can be determined to implement the protocol with the according statistical security parameter.

To guarantee the randomness of b, it can be checked with a sacrifice. However, this may leak some bits of $\mathbf{a}$ if a malicious party used inconsistent inputs in some of the OTs before. This is the reason why inner products are used. All parties sample a random value $r \in \mathbb{F}^\tau$ and obtain the triple (a, b, c) with $a = \langle a, r \rangle$ and $c = \langle c, r \rangle$. This ensures that any leaking bits of $\mathbf{a}$ are combined with not leaking bits so a appears uniformly random. The same applies to c. Since b is checked with a sacrifice, a second triple needs to be generated. Instead of repeating this step, another random r can be sampled and used to create a correlated triple with the same b.

The triple generation protocol realizes this optimized idea. It starts with the **Multiply** step which uses random oblivious transfer (ROT) to compute a secret sharing of the product $a \cdot b$. This is done by each pair of parties running τ copies of the basic two-party product sharing protocol. To clarify, for each finally created triple there are τ triples generated which will be combined to one.

doi: 10.2313/NET-2022-11-1_06

In this step a corrupt party might guess some bits of a, that is why τ components of a are used instead of only one. Since b is already uniformly random, no privacy amplification is needed here. Afterwards, each party sums up their shares to obtain an additively shared triple which can be incorrect if a malicious party was dishonest.

In the next step, the **Combine** step, the parties take random linear combinations of the τ components and two randomly sampled values $r, \hat{r} \in \mathbb{F}^\tau$. Thereby, they obtain the two triples where one will be sacrificed later. Following the **Authenticate** step adds MACs to both triples. Because the b is included in both triples, only five values need to be authenticated. Lastly, the correctness of one triple is checked in the **Sacrifice** step. Therefore, all parties first sample a random value s. They then locally compute and open $[\![p]\!] = s \cdot [\![a]\!] - [\![\hat{a}]\!]$. In this context, $[\![.]\!]$ is used to describe a share. With this, they are able to compute the left side of 2 because it is linear. By inserting p and transforming it comes clear that it needs to open up to zero in order for the triples to be correct.

$$s \cdot [\![c]\!] - [\![\hat{c}]\!] - [\![b]\!] \cdot p = [\![s \cdot (c - a \cdot b) + (\hat{a} \cdot b - \hat{c})]\!] \quad (2)$$

With this protocol, the adversary has no chance to cheat. Starting with the **Multiply** stage, any nonzero errors will be detected by the share of the random honest party. This results in an incorrect triple with a high probability. Another approach for the adversary is to guess some bits of a. If all guesses succeed, the triple can be correct and will pass the sacrifice. Thus the adversary learns the bits that were guessed which is called a selective failure attack. However, this is made more difficult by the **Combine** step. To guess a single bit of the final computed share of a, they must guess many bits of the initially generated **a** which is very unlikely to happen. Finally, the **Sacrifice** stage checks the triple with a random value which is unknown when the triples are authenticated. Therefore, it can only pass with a probability of $\frac{1}{|\mathbb{F}|}$ if the triple is incorrect.

Complete preprocessing. By securely generating triples, the main goal of the preprocessing stage was achieved with this protocol. In addition to this, it should also produce random shared values to allow the parties to provide inputs in the online phase. The party simply creates an authenticated additive share of a random value. When later inputting a value, the party broadcasts the difference between the input and the random shared value so that the other parties can adjust their share.

4. Online Phase

The online phase of the MASCOT protocol is quite forward. To share an input x, the party takes a preprocessed random value $[\![r]\!]$ and computes $x - r$. The random value works as a one-time pad and perfectly masks the secret input since it is unknown. After broadcasting the result, all parties compute $[\![r]\!] + (x - r)$ to obtain a share of x.

The multiplication is based on Beaver's circuit randomization technique. Using the multiplication triple (a, b, c) it is straightforward to multiply two secret-shared values $[\![x]\!]$ and $[\![y]\!]$. For this the values $\epsilon = x - a$ and $\rho = y - b$ are computed and opened, where the triple masks the input perfectly as it is uniformly random. Now the sharing of $x \cdot y$ can be computed locally with:

$$[\![z]\!] = [\![c]\!] + \epsilon \cdot [\![b]\!] + \rho \cdot [\![a]\!] + \epsilon \cdot \rho$$

To output a share, all previously opened input values are checked. Then the share is opened and verified through the check. If any check fails, the protocol aborts and informs all parties that no value could be computed. Since most computation was moved to the preparation phase, the amount of communication in the online phase is quite small. The only values sent in this phase of the protocol are masked openings for multiplications and outputs. Compared to other implementations, the time for a single multiplication is 200 times faster [1].

5. Conclusion

MASCOT makes faster secure computation of general arithmetic circuits possible. By computing the multiplication triples in the preprocessing phase it allows for a fast online phase without heavy computation. Moreover, through the arithmetic view of oblivious transfer, the COPE protocol succeeds to create sharings of products for the triples. Their generation is based on oblivious transfer extensions where OTs can be realized with fast, symmetric cryptography after a one-time setup. In combination with information-theoretic MACs, it authenticates the triples immediately after creation and when later opening values. By sacrificing another triple the correctness of the multiplication triple is verified. Through the consistency checks and with the sacrifice technique it achieves active security against up to $n-1$ corrupted parties considering a dishonest majority.

In conclusion, the MASCOT protocol improves the SPDZ protocol in the preprocessing phase. Through the reduced computation and communication, it is applicable in the real world by still ensuring active security. It is the first protocol that is making use of oblivious transfer to generate the multiplication triples.

References

[1] M. Keller, E. Orsini, and P. Scholl, "Mascot: Faster malicious arithmetic secure computation with oblivious transfer," https://eprint.iacr.org/2016/505.pdf, 2016, [Online; accessed 07-April-2022].

[2] P. K. Madhusudan L, "Information-theoretic macs," https://www.csa.iisc.ac.in/~arpita/Cryptography15/CT4.pdf, 2015, [Online; accessed 07-April-2022].

[3] D. Beaver, "Correlated pseudorandomness and the complexity of private computations," *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pp. 479–488, 1996.

[4] N. Gilboa, "Two party rsa key generation," *Advances in Cryptology - CRYPTO*, pp. 116–129, 1999.

[5] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," *Advances in Cryptology - CRYPTO*, pp. 145–161, 2003.