TinyOLE: Efficient Actively Secure Two-Party Computation from Oblivious Linear Function Evaluation

Nico Döttling
Friedrich-Alexander-University Erlangen-Nürnberg
nico.doettling@gmail.com

Satrajit Ghosh*
Aarhus University
satrajit@cs.au.dk

Jesper Buus Nielsen†
Aarhus University
jbn@cs.au.dk

Tobias Nilges*
Aarhus University
tobias.nilges@cs.au.dk

Roberto Trifiletti†
Aarhus University
roberto@cs.au.dk

ABSTRACT

We introduce a new approach to actively secure two-party computation based on so-called oblivious linear function evaluation (OLE), a natural generalisation of oblivious transfer (OT) and a special case of the notion of oblivious polynomial evaluation introduced by Naor and Pinkas at STOC 1999. OLE works over a finite field $\mathbb{F}$. In an OLE the sender inputs two field elements $a \in \mathbb{F}$ and $b \in \mathbb{F}$, and the receiver inputs a field element $x \in \mathbb{F}$ and learns only $f(x) = ax + b$. Our protocol can evaluate an arithmetic circuit over a finite field $\mathbb{F}$ given black-box access to OLE for $\mathbb{F}$. The protocol is unconditionally secure and consumes only a constant number of OLEs per multiplication gate. An OLE over a field $\mathbb{F}$ of size $O(2^\kappa)$ can be implemented with communication $O(\kappa)$. This gives a protocol with communication complexity $O(|C|\kappa)$ for large enough fields, where C is an arithmetic circuit computing the desired function.

This asymptotically matches the best previous protocols, but our protocol at the same time obtains significantly smaller constants hidden by the big-O notation, yielding a highly practical protocol. Conceptually our techniques lift the techniques for basing practical actively secure 2PC of Boolean circuits on OT introduced under the name TinyOT by Nielsen, Nordholt, Orlandi and Burra at Crypto 2012 to the arithmetic setting. In doing so we develop several novel techniques for generating various flavours of OLE and combining these.

We believe that the efficiency of our protocols, both in asymptotic and practical terms, establishes OLE and its variants as an important foundation for efficient actively secure 2PC.

*Supported by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant #669255 (MPCPRO).
†Supported by the European Union's Horizon 2020 research and innovation programme under grant agreement #731583 (SODA).

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134024

CCS CONCEPTS

• **Theory of computation** → **Cryptographic protocols**; **Communication complexity**; • **Security and privacy** → *Information-theoretic techniques*;

KEYWORDS

Two-Party Computation, Constant Computation, Constant Communication, OLE, UC-Security

1 INTRODUCTION

Secure two-party computation (2PC) allows two distrusting parties to compute any function of their joint input without revealing any extra information about their private inputs other than the correct output. The more general case with more than two parties is called multiparty computation (MPC), but we will focus on the two party case in this work.

Our Contribution in Brief

We introduce a new approach to actively secure 2PC based on so-called oblivious linear function evaluation (OLE), a natural generalisation of oblivious transfer (OT) and a special case of the notion of oblivious polynomial evaluation introduced by Naor and Pinkas [21]. The protocol is conceptually simple, provides unconditional UC-security and uses the OLE primitive as black-box.

In a 1-out-of-2 OT the sender inputs two messages x_0 and x_1 and the receiver inputs a choice bit b and learns only x_b. OLE works over a finite field $\mathbb{F}$, where the sender inputs two field elements $a \in \mathbb{F}$ and $b \in \mathbb{F}$ and the receiver inputs a field element $x \in \mathbb{F}$ and learns only $f(x) = ax + b$. Note that if we set $b = x_0$ and $a = x_1 - x_0$, then $f(0) = x_0$ and $f(1) = x_1$. So, for the field with two elements, an OLE is equivalent to a 1-out-of-2 OT of bits. Hence an OLE can be seen as a generalisation of OT to the case of larger fields. One can efficiently implement OLE under a number of assumptions both with standalone security and UC-security [13, 17]; even with information-theoretic UC-security using tamper proof tokens [9]. Note that the protocols in [13, 17] are based on the Naor-Pinkas assumption that a noisy Shamir secret-sharing is pseudo-random, which basically assumes that one cannot do better than the standard list-decoding algorithms (where the latter do much more than break pseudo-randomness). In our efficiency comparison the parameters

in the assumption are set fairly aggressively and it is not clear if the problem has been sufficiently crypt-analysed.

Our protocol can evaluate an arithmetic circuit over a finite field $\mathbb{F}$ given black-box access to OLE for $\mathbb{F}$. The approach consumes only 22 OLE per multiplication gate. An OLE over a field $\mathbb{F}$ of size $O(2^\kappa)$ can be implemented with communication $O(\kappa)$, where κ is the security parameter. This gives a protocol with communication complexity $O(|C|\kappa)$ for large enough fields, where C is an arithmetic circuit computing the desired function. This asymptotically matches the best previous protocols, but our protocol at the same time obtains significantly smaller constants hidden by the big-O notation.

Conceptually our techniques lift the techniques by Nielsen *et al.* in [22] for basing practical actively secure 2PC of Boolean circuits on OT to the arithmetic setting (known as the TinyOT protocol). In doing so we develop several novel techniques for generating various flavours of OLE and combining these.

Related Work

The first protocol to realize 2PC was introduced by Yao [23]. In this protocol, a sender garbles all gates in a Boolean circuit computing the desired function and with the input of the sender hard wired, sends the garbled circuit to the receiver who learns the garbled version of his input for each of his input wires via OT. The receiver can then evaluate the circuit on his garbled input and obtain only the output. The protocol requires that the garbler constructs a correctly formed garbled circuit and is therefore only secure against a semi-honest garbler. A semi-honest party follows the prescribed protocol, but tries to learn extra information from their view in the protocol. Garbling one Boolean gate requires a small number of evaluations of a hash function or symmetric cipher and sending a small number of outputs of the hash function or cipher. The communication complexity is therefore $O(|C|\kappa)$ where κ is the size of a hash value or cipher text.

For most realistic scenarios, semi-honest security is not enough. Goldreich, Micali and Wigderson [14] proposed a new semi-honest secure 2PC protocol based on OT and they also presented a general transformation that ensures semi-honest behaviour even for active adversaries that might deviate from the protocol. The drawback is that to achieve active security, they have to make extensive use of zero-knowledge proofs, thus rendering the protocol too inefficient for practical purposes.

There have been a number of works trying to improve the asymptotic efficiency of actively secure two-party computation and more recently also many works implementing 2PC to benchmark the practical performance of various approaches to 2PC. In the two party case most works following the seminal papers of [14, 23] focused on the garbled circuit approach and have yielded considerable improvements over the earlier protocols. The amount of work on efficient garbling is overwhelming and since our focus here is on the OT based line of work, we have chosen to only mention some of the latest work on garbled circuits.

Notably, Lindell shows in [19] how to get active security except with probability 2^{-s} at the price of s times the work of the semi-honest garbling protocol using the so-called cut-and-choose approach where the garbler produces s garbled circuits of which some are opened up for a correctness check and the remaining are used for evaluation. This is secure as long as we are not unlucky to open exactly the correct garbled circuits and use exactly the incorrect ones for evaluation. Since each circuit is selected for test uniformly at random this gives the claimed security of 2^{-s}. Here s is a statistical security parameter separate from the computational security parameter.[1] The complexity of the protocol is $O(|C|\kappa s)$, where κ is the length of the output of a symmetric primitive like a hash function or symmetric cipher.

In [11] Frederiksen *et al.* presented a different garbling approach where many individual gates are garbled, a constant fraction opened for a correctness check and the rest stitched together to a robust circuit for computing the desired function. For large enough circuits the asymptotic complexity of this approach is $O(\kappa|C|s/\log(s))$, which is asymptotically better than [19] but less efficient for meaningful levels of security because of the large constants hidden by the big-O notation.

In [20] Lindell and Riva synthesised the two approaches mentioned above for the case of evaluating the same function many times. The asymptotic complexity is still $O(\kappa|C|s/\log(s))$, but the constants are much smaller than in [11]. This appears to be the most practical approach to actively secure garbling-based 2PC to date, albeit for the specialised setting of computing the same function many times.

There has also been significant progress on OT-based 2PC. The semi-honest protocol in [14] is fairly simple. There are two parties denoted P_1 and P_2. Each bit b in the computation is represented by two bits b_1 and b_2, where b_1 is uniformly random and $b_2 = b \oplus b_1$ such that $b = b_1 \oplus b_2$ and where b_i is known only to P_i. Given two bits a and b represented like this the parties can securely compute a representation of $c = a \oplus b$ by letting P_i compute $c_i = a_i \oplus b_i$. We can reveal a bit a to party P_1 by having P_2 send a_2 and we can similarly reveal a bit to P_2. Since exclusive or and conjunction are universal for Boolean computations, it is therefore sufficient to be able to securely compute a representation of $c = ab$ from representations of a and b. This is slightly more tricky. We use that $ab = a_1b_1 \oplus a_1b_2 \oplus a_2b_1 \oplus a_2b_2$, where P_i can compute a_ib_i. The only troublesome terms are therefore a_1b_2 and a_2b_1. If we can compute secure representations of these terms we are done as we already know how to securely compute exclusive or. For each of the troublesome terms we will use one OT. Let us compute a representation of a_1b_2 as an example. Party P_1 samples a uniformly random bit c_1 and offers the following two messages in the OT: $x_0 = c_1$ and $x_1 = c_1 \oplus a_1$. Then P_2 selects the message x_{b_2} and sets $c_2 = x_{b_2}$. It is easy to see that $x_{b_2} = c_1 \oplus a_1b_2$. Hence $c_1 \oplus c_2 = a_1b_2$, as desired. This protocol is fairly efficient as it consumes only two OTs per conjunction gate in the circuit, i.e., the complexity is $O(|C|\kappa)$ if we use κ to denote the complexity of an OT. It is, however, easy to see that the protocol is not actively secure. We asked P_1 to input $x_0 = c_1$ and $x_1 = c_1 \oplus a_1$ to the OT. She might instead offer $x_0 = c_1 \oplus 1$ and $x_1 = c_1 \oplus a_1 \oplus 1$ which can be used to flip the bit on an internal wire of the circuit, which might give a wrong result and hence leak unintended information, hence the need for zero-knowledge to prove that the right values were input.

[1] The definition of a statistical security parameter s is that if we fix s and let the computational security parameter κ grow, then the security of the scheme will go to 2^{-s} faster than any inverse polynomial in κ.

In 1995 Crépeau, van de Graaf and Tapp [6] introduced the notion of committed oblivious transfer (COT) which is a variant of OT where the parties are committed to inputs and outputs of the OT using homomorphic commitments. This allows to open the difference between the output of one OT and the input of another OT to prove that they are the same. Thus the parties are forced to input the right values to the OTs which ensures active security. They show how to implement one COT given blackbox access to $O(s)$ OTs, where the big-O notation hides some moderately large constants. This gives a protocol with complexity $O(|C|\kappa s)$, where κ is the complexity of an OT.

In 2008 Ishai, Prabhakaran and Sahai [16] introduced a radically different approach to OT-based 2PC known as the IPS-compiler. The two parties will use a semi-honest OT-based 2PC to simulate a large number of virtual parties and use a special version of cut-and-choose known as the watchlist mechanism. This allows to inspect some randomly selected ones of these virtual parties to check if they are simulated according to the semi-honest protocol. If so, then most of them must indeed have been run correctly. These virtual parties will then run between them an asymptotically efficient secure multi-party protocol for computing the desired function secure against a minority of actively corrupted parties. Such MPC protocols exist with complexity $O(|C|)$. The resulting 2PC protocol therefore has complexity $O(|C|\kappa)$, where κ is the complexity of one OT. The hidden constants resulting from creating the virtual parties and the constant suffered by the best asymptotically good MPC protocols like [7], however, results in fairly large constants being hidden by the big-O notation.

In 2012 Nielsen *et al.* [22] then published a new approach to OT based 2PC, later called TinyOT. They essentially show how to construct actively secure COT from actively secure OT by consuming only $O(s/\log(s))$ OTs per COT and with very small constants hidden by the big-O notation. In their protocol they first produce in an offline phase a number of COTs on random values and then in the online phase use that COT is random self reducible to run an actively secure version of [14] using COT instead of OT. One COT on random values is produced simply by running one OT on random values and then letting the parties commit to their inputs and outputs. Then an efficient test is run on each such potential COT to verify that the commitments were computed correctly. The commitments and the test consume a small constant number of extra OTs. Unfortunately the test has the property that the sender might cheat in the test itself and use it to verify a guess at the random choice bit of the receiver. If the guess is wrong, the sender is detected in her cheating, but if the guess is correct the cheating is undetected. This selective error attack means that the sender can undetectably guess a total of s bits with probability 2^{-s}. By setting s large enough that 2^{-s} is negligible and preparing for instance s^4 COTs it can however be guaranteed that at most a fraction s^{-3} of these had the choice bit leaked. To get rid of these relatively few bad COTs a COT combiner is used. This is a primitive which combines B COTs of which at most $B-1$ are bad into one fully secure COT. By randomly grouping the s^4 COTs into buckets of size B a given bucket gets filled with bad COTs only with probability $(s^{-3})^B$, so to get security 2^{-s} one needs buckets of size just $B = s/3\log_2(s)$, which for a practical statistical security parameter like $s = 64$ would be just $B = 4$. At the same time it was shown in [22] how to do efficient actively secure OT extension, improving on the semi-honest secure OT extension from [15]. Specifically they show how to use κ actively secure OTs, where κ is the security parameter, to produce any polynomial number of actively secure OTs by using only a small number of additional applications of a hash function per produced OT. This quickly amortizes away the cost of the κ seed OTs and essentially means the prize of an actively secure OT is a small constant number of applications of a hash function. This overall yields a protocol with sub-optimal asymptotic complexity of $O(|C|\kappa s/\log_2(s))$, but with $s/\log_2(s)$ and the hidden constants so small in practice that it yielded the most practical protocol for actively secure 2PC at the time. The protocol was implemented and benchmarking times reported in [22] indicate that the approach can be practical for reasonably large circuits.

A common drawback of the garbling and OT based approaches is that they work with Boolean circuits. This means that if you want to securely compute an arithmetic circuit over a large field with $|\mathbb{F}| = O(2^\kappa)$ and you implement the arithmetic operations by small Boolean sub-circuits, then the communication complexity of [16] and [22] will be at least $O(|C|\kappa^2)$ and $O(|C|\kappa^2 s/\log_2(s))$, where $|C|$ is the number of gates in the arithmetic circuit. This gives a poor asymptotic efficiency. However, in a later improvement and extension of the TinyOT approach, called MASCOT [18], Keller, Orsini and Scholl showed that OT based protocols can in fact be relatively efficient in practice for doing arithmetic computation. It is hard to compare our new protocol to MASCOT as the comparison depends on the size of the field and the security parameter. However, we will compare our protocol to the MASCOT protocol in Section 6 for some concrete parameters and see that our protocol compares favourably to MASCOT for most field sizes and security parameters. Since MASCOT is an improved version of TinyOT, our new protocol therefore also beats TinyOT.

There is a line of work on implementing actively secure MPC of arithmetic circuits, including [3, 5, 8]. These protocols all use some form of homomorphic encryption along with zero-knowledge proofs for proving correct behaviour. The complexity of these protocols when run between two parties is $O(|C|\kappa)$, where κ is the size of a ciphertext of the homomorphic encryption scheme. These protocols are all designed for the *multi*party setting and not the 2PC setting, and though they can all be run between two parties they are not well suited for this setting, i.e., they are not competitive with existing 2PC solutions because of the relatively large cipher texts of homomorphic encryption schemes and the overhead of the zero-knowledge proofs. In particular, the reported timings of the implementations are slower than for instance [22].

The works [17] and [12] are more directly comparable to our protocol. In [17] Ishai, Prabhakaran and Sahai instantiate the above mentioned IPS compiler to get an arithmetic 2PC protocol with communication complexity $O(G\kappa)$ for evaluating an *arithmetic* circuit with G gates. If the size of the field in which the arithmetic is done is $\Theta(2^\kappa)$, this gives a protocol with constant communication overhead in the following sense: the communication complexity of the protocol is within a constant of the communication needed to send a trace of an evaluation of the arithmetic circuit. In [12] Genkin *et al.* show how to get a similar result by compiling passive secure 2PC protocol into active secure 2PC protocols using circuits

secure against active attacks. These works are pinnacles on the theoretical work on active-secure arithmetic 2PC and gives very general and modular ways to build protocols with the same asymptotic complexity as ours. As a result of the general approaches, however, these protocols hide large constant using the big-O notation. In particular, there seem to be no attempt in the literature to work out the exact complexity of protocols based on a given instantiation. Compared to [12, 17], our work sells a lot of the generality by focusing on a concrete primitive, namely OLE, but at the same times gains considerably in concrete efficiency by being able to design protocols targeted for this particular primitive.

It is difficult to provide a fair comparison with [12, 17] due to their generality, as the efficiency of the approaches depend on how they are instantiated. It is hard to ensure that all possible instantiations have been considered and that none are more efficient than our approach. That being said, [12] uses basically the same semi-honest approach as we do, and then adds AMD codes, MACs and OLEs to obtain malicious security. Setting aside the cost to authenticate inputs, each multiplication gate only requires 6 OLEs. However, this does not take into account the AMD encoding and decoding circuit, which is used in *every* gate. While these circuits are constant in size, they are dependent on the AMD code and therefore it is difficult to argue about their characteristics. But, if the AMD encoding and decoding circuits combined require as few as 3 multiplications, the total number of OLEs per multiplication gate is already 24, which is less efficient than our new approach. In addition every addition gate would also require OLE calls. Another point is that to the best of our knowledge, in the envisioned application, the AMD code would entail a multiplicative communication overhead of 3, which further reduces the communication efficiency.

The paper [17] uses only passively secure OLE (which saves a factor of 2 in computation and communication over the currently most efficient protocol for actively secure OLEs in [13]). The problem then is finding a suitable outer protocol for the IPS compiler. A straight-forward approach is to use as the outer protocol a protocol which computes many OLEs in parallel, as the most efficient MPC protocols are protocol which compute the same function many times in parallel. The generated actively secure OLEs can then be used for instance by our new protocol. It seems as the best choice to use a version of [2] where packed secret-sharing is used to compute many OLEs in parallel and where one removes all the machinery in that paper concerned with guaranteed termination. This can be done as the output of the compiler is a two-party protocol for which termination is not guaranteed anyway. Even then, using [2] as the outer protocol, one will get an overhead of at least 4 in the number of OLEs and therefore in communication. As a consequence each produced malicous-secure OLE is twice a expensive as using [13]. This completely ignores the setup of the watchlists etc. Another approach is to use a packed outer protocol to compute many instances of some other gadgets which can be used to compute a single circuit. An obvious choice is to generate authenticated multiplication triples as in Fig. 6. Each triple requires 6 authenticated values and one multiplication. Each authenticated value in itself requires one multiplication, for a total of 7 multiplications. With the liberally estimated overhead of 4 from above, this gives 28 multiplications on the outer protocol. Each of these should be emulated by an inner protocol where each multiplication costs at least 2 semi-honest OLEs, for a total of 56 semi-honest OLEs. Using [13] one actively secure OLE can be generated for the price of 2 semi-honest OLEs. So the price in actively secure OLEs would be around 28. This is still not competitive, but is close enough that it is an interesting open question whether [17] can be used to beat our protocol via computing many instances of some small gadget appropriate for actively secure 2PC. A final approach is to use as the outer protocol a protocol like [7] which directly computes a single function. However, [7] has a poly-logarithmic (any non-constant) overhead and also seem to have a huge concrete overhead for practical security levels. This approach therefore does not seem to be a practicable 2PC based on OLE.

Our Contributions

We introduce a new approach to actively secure 2PC which can securely compute an arithmetic circuit C over any field while using black-box access to only a very small constant number of OLEs over $\mathbb{F}$ per arithmetic gate in C. Specifically we use 22 OLE per multiplication gate in C, while at least 8 OLE are necessary to compute an authenticated multiplication even with semi-honest security. Additionally, the parties only have to perform a constant number of additions and multiplications. Since one OLE over many fields can in turn be implemented with communication $O(\kappa)$, where κ is the security parameter, under standard assumptions this gives a protocol with communication $O(|C|\kappa)$. This means the protocol is asymptotically as good as [16] when the field size is large, but the concrete complexity is much better. At the same time, the number of OLEs used by our protocol per *arithmetic* gate is significantly less than the number of OTs used by [22] per *Boolean* gate.

The only previous protocol we are aware of which achieves the same communication complexity as our protocol using black-box access to a general assumption is [17]. In [17] they show how to evaluate an arithmetic circuit C while using a constant number of black-box accesses to homomorphic encryption per gate. This gives a complexity of $O(|C|\kappa)$, where κ is the size of a ciphertext. The protocol, however, goes via the general framework in [16] and as such the big-O notation hides some fairly large constants. Our approach can therefore be seen as combining the good asymptotic complexity of [17] with the same good practical efficiency of [22].

Our Techniques

Our approach is somewhat reminiscent of the approach in [22]. We start with an arithmetic version of [14]. A value $a \in \mathbb{F}$ is represented by $a = a_1 + a_2$ where a_2 is uniformly random and a_i is known only by P_i. Secure addition is straight forward: $c_i = a_i + b_i$. To securely compute a representation of $c = ab$ one again uses that $ab = a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2$. To compute a secure representation of a troublesome term like $c = a_1b_2$ use one OLE. Let the sender P_1 of the OLE pick uniformly random c_1 and input $(a, b) = (a_1, -c_1)$ and let the receiver input $x = b_2$. Then they invoke the OLE to compute $c_2 = ax + b = a_1b_2 - c_1$. Clearly $c_2 + c_1 = a_1b_2$.

We then introduce a protocol to batch-authenticate random values. Using this protocol in an offline phase allows us to prepare authenticated values for later use in the online phase, but it is also a crucial building block in order to generate authenticated multiplication triples. Instead of using a cut-and-choose approach and

bucketing or randomness extraction to ensure consistent inputs for the MACs of the values, we design a simple variant of OLE where one party can input a scalar, while the other party inputs vectors of field elements[2]. Our protocol has a small additive overhead, which is independent of the number of values that are authenticated, and can be build black-box from OLE. This immediately saves us an asymptotic factor of $O(s/\log(s))$ over [22]. What is more, this protocol simplifies the analysis of the authenticated multiplication triples, since the checks can be designed such that they test the inputs against the already verified authenticated values.

These checks are significantly different from the test in [22] which used a hash function. We only use black-box access to OLE. Furthermore, we manage to work around the selective error problem encountered in [22]. Our checks in principle have a similar problem with a selective attack, but we can carefully arrange the checks such that the sender of a and b needs to guess the random field element x to not get detected. This immediately gives security $|\mathbb{F}|^{-1}$, which is negligible for large enough fields. Developing the new tests required developing significantly new techniques which exploit essentially that we are in the arithmetic setting where both parties have a large random input. At the same time our checks are simple enough that we get a very practical protocol.

Unfortunately our new techniques does not translate back to the OT based protocol from [22], as there the value corresponding to our x is the choice bit of the receiver which can of course be guessed with good probability. This shows that an essential prerequisite for our efficiency improvement is moving to the arithmetic setting with a large field, and we indeed exploit the arithmetic structure of the OLE primitive in many places throughout our protocols.

We believe that the efficiency of our protocols, both in asymptotic and practical terms, establishes OLE and its variants as an important foundation for efficient actively secure arithmetic 2PC.

2 PRELIMINARIES

2.1 Notation

By $\mathbb{F}_q$ we denote a finite field of size q, $\mathbf{x}$ denotes a vector of field elements, while x denotes a single field element. By $a \mathrel{+}= b$ we denote $a = a + b$.

The notation for our authenticated values is depicted in Figure 1. By M_{v_x} (or K_{v_x}) we denote the MAC (resp. key) associated with the name v_x, not its value.

2.2 UC Framework

We state and prove our results in the Universal Composability (UC) framework of Canetti [4]. In the framework, security of a protocol is shown by comparing a real protocol π in the real world with an ideal functionality $\mathcal{F}$ in the ideal world. $\mathcal{F}$ is supposed to accurately describe the security requirements of the protocol and is secure per definition. An environment $\mathcal{Z}$ is plugged either to the real protocol or the ideal protocol and has to distinguish the two cases. For this, the environment can corrupt parties. To ensure security, there has to exist a simulator in the ideal world that produces a protocol transcript indistinguishable from the real protocol, even if the environment corrupts a party. We say π UC-realizes $\mathcal{F}$ if for all adversaries $\mathcal{A}$ in the real world there exists a simulator $\mathcal{S}$ in the ideal world such that all environments $\mathcal{Z}$ cannot distinguish the transcripts of the parties' outputs.

For our constructions we assume active adversaries and static corruption and achieve statistical indistinguishability. We use some simplifications in the descriptions of the ideal functionalities in order to enhance the readability,

[2]Such a primitive was recently introduced as vector OLE in [1].

Global Key We call $\Delta_A, \Delta_B \in \mathbb{F}_q$ the two *global keys*, held by B and A, respectively. These values are uniformly random and not known by the other party.

Authenticated Value $[x]_A$ represents an *authenticated secret value* held by A. B holds the corresponding key $K_x \in \mathbb{F}_q$, while A holds $x \in \mathbb{F}_q$ and a MAC $M_x = K_x + x\Delta_A \in \mathbb{F}_q$.

Let $[x]_A = (x, K_x, M_x)$. We omit Δ_A since it is identical for each value x. To compute $[z]_A = [x]_A + [y]_A$, A computes $[z]_A = (z, K_z, M_z) = (x+y, K_x+K_y, M_x+M_y)$. Here, A locally computes z and M_z, while B locally computes K_z.

To authenticate a constant value $v \in \mathbb{F}_q$ known to both parties, A sets $M_v = 0$ and B sets $K_v = v\Delta_A$. Then $[v]_A = (v, K_v, M_v)$. For a constant c we let $[x]_A + c = [x]_A + [c]_A$ and $[x]_A \cdot c = [xc]_A = (xc, K_x c, M_x c)$.

We say A *reveals* $[x]_A$ by sending (x, M_x) to B, who aborts if $M_x \neq K_x + x\Delta_A$. We say A *announces* x by sending x to B without the MAC.

Authenticated Share We let $[x] = [x_A|x_B]$ denote that A and B hold authenticated shares x_A, x_B such that $x = x_A + x_B$.

To compute $[z] = [z_A|z_B] = [x]+[y]$, A locally computes $z_A = x_A + y_A$ and B locally computes $z_B = x_B + y_B$.

An authenticated share on a constant value $c \in \mathbb{F}_q$ can be obtained by setting $[c] = [c|0]$. We let $c[x] = [cx]$.

To *reveal* an authenticated share, the parties reveal the authenticated values and abort if the MACs are not correct.

Figure 1: Notation of authenticated values and shares.

2.3 Arithmetic Circuits

An arithmetic circuit C over the field $\mathbb{F}$ consists of addition and multiplication gates. Each such gate has a fan-in of 2, i.e., takes as input 2 field elements a and b and computes $a+b$ or $a \cdot b$, respectively. These inputs can either be variables or constants.

2.4 Oblivious Linear Function Evaluation

Oblivious Linear Function Evaluation (OLE) is a special case of Oblivious Polynomial Evaluation (OPE). In contrast to OPE, only linear functions can be obliviously evaluated. A party A has as input two values $a, b \in \mathbb{F}_q$ that determine a linear function over $\mathbb{F}_q$, and a party B wants to obliviously evaluate the linear function on input $x \in \mathbb{F}_q$ (cf. Figure 2).

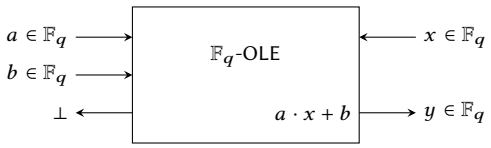

Figure 2: The $\mathbb{F}_q$-OLE primitive.

There are several implicit and explicit constructions of OLE based on a variety of assumptions, e.g. [8, 9, 13, 18]. For our protocol we assume OLE as a (UC-secure) black-box.

3 TOOLS

In this section we present the building blocks that are later needed for the 2PC protocol. The first protocol shows how to construct $\mathbb{F}_q^k$-OLE from $\mathbb{F}_q$-OLE with a small additive overhead.

3.1 Commitments from OLE

We will later need commitments in our protocols. Towards this, Döttling *et al.* [9] present a very simple UC-secure protocol for commitments from OLE (cf. Figure 3). We state it here for completeness.

Protocol Π_{Com}

Commit Phase: The sender gets as input s and chooses a random value $b \in \mathbb{F}_q$ and input (s, b) into $\mathbb{F}_q$-OLE. The receiver then chooses a random value $x \in \mathbb{F}_q$ and inputs it into $\mathbb{F}_q$-OLE to obtain y.

Unveil Phase: The sender sends (s, b) to the receiver. If $s \cdot x + b = y$ the receiver accepts, otherwise he aborts.

Figure 3: Commitment protocol from $\mathbb{F}_q$-OLE.

Lemma 3.1 ([9]). *Protocol Π_{Com} UC-realizes $\mathcal{F}_{\mathsf{Com}}$ in the $\mathbb{F}_q$-OLE-hybrid model.*

Against a corrupted committer, the simulator can directly extract the input into $\mathbb{F}_q$-OLE, because the committer cannot learn the output of $\mathbb{F}_q$-OLE and therefore cannot change his input afterwards. Towards security against a corrupted receiver, the simulator can first commit to a random value r, but he learns the challenge x. This allows to find a value b' which provides a correct derandomization. See [9] for more details.

3.2 Constant Rate Dimension Extension for OLE

It will be convenient for us to have a tool that enforces the receiver to use a single input in several OLE instances. While the receiver inputs a field elements from $\mathbb{F}_q$, the sender inputs vectors from $\mathbb{F}_q^k$. $\mathbb{F}_q^k$-OLE computes $y_i = a_i \cdot x + b_i$ for all $i \in \{1, \ldots, k\}$, i.e. uses the same receiver input for all OLE-instances. The primitive is depicted in Figure 4.

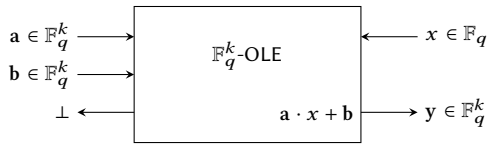

Figure 4: The $\mathbb{F}_q^k$-OLE primitive.

Döttling *et al.* [10] show how to achieve this with an overhead of a small multiplicative constant of $2 + \varepsilon$, but their construction is more general in the sense that it works for any field $\mathbb{F}_q$. In our scenario, we only consider fields that have size exponential in the security parameter. Our reduction has only an *additive* constant overhead of 2 $\mathbb{F}_q$-OLE instances.

Consider the protocol in Figure 5. The sender has k pairs of random inputs (a_i, b_i), while the receiver has one random input x. The main idea is to use one additional OLE to check that the receiver uses the same input in all OLE as specified by the protocol. The sender chooses one additional input $a_{k+1} = \sum_{i=1}^k a_i$, while the receiver chooses $\mathbf{r} \in \mathbb{F}_q^{k+1}$ uniformly at random. The parties execute $k + 1$ $\mathbb{F}_q$-OLE, but from receiver to sender, i.e. the receiver inputs (x, r_i) and sender a_i. Let $t_i = a_i x + r_i$ be the result obtained by the sender. If both parties behaved according to the protocol, then

$$\sum_{i=1}^{k} t_i - t_{k+1} = \gamma = \sum_{i=1}^{k} r_i - r_{k+1}, \tag{1}$$

where the sender locally computes the LHS of Equation (1) and the receiver the RHS of Equation (1). Then they check the equality of both values. For this, the sender first commits to his value, then the receiver sends his value to the sender, who checks the equality and provides the decommitment. If the decommitment is correct, the sender sends $t_i + b_i$ to the receiver, who removes his blinding value r_i to get $a_i x + b_i$. This commitment can again be realized using an OLE (cf. Section 3.1), so that the total overhead for $\mathbb{F}_q^k$-OLE is 2 $\mathbb{F}_q$-OLE instances. Note that after the check is verified, the values can be derandomized with standard techniques.

Protocol $\Pi_{\mathbb{F}_q^k\text{-OLE}}$

Let COM be the commitment from Section 3.1.

Receiver: Pick random $x \in \mathbb{F}_q$, $\mathbf{r} \in \mathbb{F}_q^{k+1}$. Input (x, r_j) into OLE j for $j \in [k+1]$.

Sender: Pick random $\mathbf{a} \in \mathbb{F}_q^k$, $\mathbf{b} \in \mathbb{F}_q^k$. Set $a_{k+1} = \sum_{i=1}^k a_i$ and input a_j into OLE j. Obtain $\mathbf{t} = \mathbf{a}x + \mathbf{r}$ and compute $\gamma = t_{k+1} - \sum_{i=1}^k t_i$. Further compute $(\mathsf{com}, \mathsf{unv}) \leftarrow \mathsf{COM.Commit}(\gamma)$ and $u_i = t_i + b_i$ for $i \in [k]$. Send $(\mathsf{com}, \mathbf{u})$ to the receiver.

Receiver: Compute $\gamma' = \sum_{i=1}^k r_i - r_{k+1}$ and send it to the sender.

Sender: If $\gamma \neq \gamma'$, abort. Send unv to the receiver.

Receiver: Check if $\mathsf{COM.Open}(\mathsf{com}, \mathsf{unv}, \gamma') = 1$ and abort if not. Set $y_i = u_i - r_i = a_i x + b_i$, $i \in [k]$ and output $\mathbf{y}$.

Figure 5: Reduction of $\mathbb{F}_q^k$-OLE to $\mathbb{F}_q$-OLE.

Lemma 3.2. *The protocol $\Pi_{\mathbb{F}_q^k\text{-OLE}}$ in Figure 5 UC-realizes $\mathbb{F}_q^k$-OLE in the $\mathbb{F}_q$-OLE hybrid model.*

Sketch. **Corrupted Sender.** Assume that $\mathcal{A}_S$ uses an input $a_{k+1} \neq \sum_{i=1}^k a_i$. The simulator extracts all inputs into the $\mathbb{F}_q$-OLE and aborts if the check is passed in this case. Since COM is binding,

from the view of $\mathcal{A}_S$ it holds that

$$\begin{aligned}\gamma &= \sum_{i=1}^{k} r_i - r_{k+1} \\ &= \sum_{i\neq j}^{k} t_i - a_i x - (t_{k+1} - a_{k+1}x) \\ &= \sum_{i=1}^{k} t_i - t_{k+1} + x(a_{k+1} - \sum_{i=1}^{k} a_i)\end{aligned}$$

Thus, γ is uniformly random over the choice of x and the check will fail in the real protocol with probability $1 - 1/|\mathbb{F}_q|$.

Corrupted Receiver. First, assume w.l.o.g. that $\mathcal{A}_R$ uses a different $x' = x + e$ in location j. The simulator extracts all inputs into the $\mathbb{F}_q$-OLE and aborts if the check is passed in this case. Since COM is hiding, from the view of $\mathcal{A}_R$ it holds that

$$\begin{aligned}\gamma &= \sum_{i=1}^{k} t_i - t_{k+1} \\ &= \sum_{i\neq j}^{k} a_i x + r_i + a_j(x+e) + r_j - (\sum_{i=1}^{k} a_i x + r_{k+1}) \\ &= \sum_{i=1}^{k} r_i - r_{k+1} + a_j e\end{aligned}$$

Thus, γ is uniformly random over the choice of a_j and the check will fail in the real protocol with probability $1 - 1/|\mathbb{F}_q|$. This argument generalizes to incorrect inputs in several positions, since each a_i is independent of the other inputs. □

Remark. If several instances of the above protocol are executed, the verification of the checks can be extended to cover all instances at once by simply adding the check values. If the check fails, the complete protocol has to be repeated, but critically a cheating party will be caught in any case.

Remark. If we want to execute batch-authentication, the roles are somewhat changed, which simplifies the protocol to a certain extend. Looking ahead, the sender holds both a global MAC key Δ and the "local" keys $\mathbf{K}$, while the receiver has a set of input values $\mathbf{v}$. The value v_{k+1} is generated as above, and K_{k+1} is chosen uniformly at random. Then the parties simply input these values into $k+1$ OLEs and compute γ as above.

4 THE DEALER PROTOCOL

In this section we will provide a protocol to generate shared and authenticated multiplication triples. We call this protocol the dealer protocol. The functionality $\mathcal{F}_{\text{Deal}}$ realized by this protocol is given in Figure 6.

The main idea behind the dealer protocol is as follows. It allows the parties to distribute (random) authenticated values and (random) authenticated multiplication triples. These precomupted values can be used in an online phase later on to perform the actual two-party computation via derandomizing these precomputed values (cf. Section 5). In principle, all values in the two-party computation will be authenticated and allow only additively homomorphic operations. Since the authentication is a multiplicative operation, the parties are bound to follow the protocol or produce an incorrectly authenticated output.

$\mathcal{F}_{\text{Deal}}$

Initialize: On input (init) from A and B, sample $\Delta_A, \Delta_B \leftarrow_\$ \mathbb{F}_q$, $\mathbf{v}_A, \mathbf{v}_B, \mathbf{K}_{\mathbf{v}_A}, \mathbf{K}_{\mathbf{v}_B} \leftarrow_\$ \mathbb{F}_q^n$ and set $\mathbf{M}_{\mathbf{v}_A} = \Delta_A\mathbf{v}_A + \mathbf{K}_{\mathbf{v}_A}$ and $\mathbf{M}_{\mathbf{v}_B} = \Delta_B\mathbf{v}_B + \mathbf{K}_{\mathbf{v}_B}$. Store $(\Delta_A, \Delta_B, \mathbf{v}_A, \mathbf{v}_B, \mathbf{K}_{\mathbf{v}_A}, \mathbf{K}_{\mathbf{v}_B}, \mathbf{M}_{\mathbf{v}_A}, \mathbf{M}_{\mathbf{v}_B})$ and output Δ_B to A and Δ_A to B.
Adversarial access: On input (setInit, A, $(\Delta_B, \mathbf{v}_A, \mathbf{M}_{\mathbf{v}_A}, \mathbf{K}_{\mathbf{v}_B})$), sample $\Delta_A \leftarrow_\$ \mathbb{F}_q$, $\mathbf{v}_B \leftarrow_\$ \mathbb{F}_q^n$ and set $\mathbf{M}_{\mathbf{v}_B} = \Delta_B\mathbf{v}_B + \mathbf{K}_{\mathbf{v}_B}$ and $\mathbf{K}_{\mathbf{v}_A} = \mathbf{M}_{\mathbf{v}_A} - \Delta_A\mathbf{v}_A$. Output Δ_A to B. Symmetrically for a malicious B.

Authenticated Value(A): On input (aValue, A) from A and B, pick an unused index i. Mark i as used and output $(\mathbf{v}[i], M_{\mathbf{v}[i]})$ to A and $(K_{\mathbf{v}[i]})$ to B.

Authenticated Value(B): Symmetrical to **AuthenticatedValue**(A)

Authenticated Multiplication: On input (aMult) from A and B, pick two unused indices i_1, i_2 and let $a_A = \mathbf{v}_A[i_1]$, $b_A = \mathbf{v}_A[i_2]$, $a_B = \mathbf{v}_B[i_1]$, $b_B = \mathbf{v}_B[i_2]$. Pick $c_A, c_B \leftarrow_\$ \mathbb{F}_q$ under the restriction that $c_A + c_B = (a_A + a_B)(b_A + b_B)$. Sample $K_{c_A}, K_{c_B} \leftarrow_\$ \mathbb{F}_q$ and set $M_{c_A} = \Delta_A c_A + K_{c_A}$ and $M_{c_B} = \Delta_B c_B + K_{c_B}$. Output $(a_A, b_A, c_A, M_{a_A}, M_{b_A}, M_{c_A}, K_{a_B}, K_{b_B}, K_{c_B})$ to A, symmetrically to B.
Adversarial access: On input (setMult, A, (c_A, M_{c_A}, K_{c_B})) pick $c_B \leftarrow_\$ \mathbb{F}_q$ under the restriction that $c_A + c_B = (a_A + a_B)(b_A + b_B)$. Set $M_{c_B} = \Delta_B c_B + K_{c_B}$ and $K_{c_A} = M_{c_A} - \Delta_A c_A$. Symmetrically for a malicious B.

Figure 6: Ideal dealer functionality.

We will start with a high level overview of our dealer protocol (cf. Figures 7 and 8). In the semi-honest case, the purpose of the dealer-protocol is to provide random triples (a_A, b_A, c_A) to A and (a_B, b_B, c_B) to B such that the relation

$$(a_A + a_B) \cdot (b_A + b_B) = c_A + c_B \tag{2}$$

holds. We can expand this to

$$a_A b_A + a_A b_B + a_B b_B + a_B b_A = c_A + c_B\ . \tag{3}$$

In the simple semi-honest protocol A chooses the values a_A and b_A at random and B the values a_B and b_B. Notice that the terms $c_A^l = a_A b_A$ and $c_B^l = a_B b_B$ can be locally computed by A and B respectively. Thus, we need to securely compute the mixed terms $a_B b_A$ and $a_A b_B$. To do so, we will introduce random offsets r_A and r_B (chosen at random by A and B respectively). We can rewrite Equation (3) as

$$(a_A b_A - r_A + a_A b_B + r_B) + (a_B b_B - r_B + a_B b_A + r_A) = c_A + c_B\ . \tag{4}$$

This suggests the following protocol to compute the triples (a_A, b_A, c_A) and (a_B, b_B, c_B):

(1) A and B choose (a_A, b_A, r_A) and (a_B, b_B, r_B) locally at random.
(2) A and B compute the values $c_A^i \leftarrow \mathbb{F}_q\text{-OLE}(a_B, r_B; b_A)$ and $c_B^i \leftarrow \mathbb{F}_q\text{-OLE}(a_A, r_A; b_B)$.
(3) A sets $c_A = c_A^l - r_A + c_A^i$ and B sets $c_B = c_B^l - r_B + c_B^i$.

Correctness of the protocol follows immediately from Equation (4). Semi-honest privacy of the protocol follows from the fact that c_A^i and c_B^i are independent of b_B and b_A and thus reveal nothing about these values (the remaining computations are local).

We will now augment this protocol into a (still semi-honestly secure) protocol such that A and B receive MACs on the other

parties' triples. The party A chooses a global MAC key Δ_B and specific keys K_{a_B}, K_{b_B} and K_{r_B}. Likewise, B chooses a global MAC key Δ_A and specific keys K_{a_A}, K_{b_A} and K_{r_A}. Now, A commits to a_A by running $M_{a_A} \leftarrow \mathbb{F}_q\text{-OLE}(\Delta_A, K_{a_A}; a_A)$. This MAC can be opened by A by sending a_A and M_{a_A} to B. The remaining MACs (M_{b_A}, M_{r_A}) and $(M_{a_B}, M_{b_B}, M_{r_B})$ are computed likewise (cf. also Figure 7).

After the above protocol is finished, A obtains MACs on the values c_A^l and c_A^i by $M_{c_A^l} \leftarrow \mathbb{F}_q\text{-OLE}(\Delta_A, K_{c_A^l}; c_A^l)$ and $M_{c_A^i} \leftarrow \mathbb{F}_q\text{-OLE}(\Delta_A, K_{c_A^i}; c_A^i)$, where $K_{c_A^l}$ and $K_{c_A^i}$ are sampled on the fly by B. Likewise, B commits to c_B^l and c_B^i by $M_{c_B^l} \leftarrow \mathbb{F}_q\text{-OLE}(\Delta_B, K_{c_B^l}; c_B^l)$ and $M_{c_B^i} \leftarrow \mathbb{F}_q\text{-OLE}(\Delta_B, K_{c_B^i}; c_B^i)$.

Now, A and B can locally compute MACs on c_A and c_B by using the additively homomorphic properties of the MAC scheme. Specifically, A computes $M_{c_A} = M_{c_A^l} - M_{r_A} + M_{c_A^i}$ and $K_{c_B} = K_{c_B^l} - K_{r_B} + K_{c_B^i}$, whereas B computes $M_{c_B} = M_{c_B^l} - M_{r_B} + M_{c_B^i}$ and $K_{c_A} = K_{c_A^l} - K_{r_A} + K_{c_A^i}$.

This concludes the description of the semi-honestly secure scheme. We now provide a mechanism to enforce semi-honest behaviour by both parties. The main ideas of this technique can be sketched as follows. To enforce that the parties input the right values in different $\mathbb{F}_q$-OLE-instances, we compute several values twice, in two different ways. We call this technique *fingerprinting*.

Since the protocol is entirely symmetric, we will describe the fingerprinting sub-protocols only from the view of A in this outline. For the first check, we will describe the generic ideas of why the check is defined as it is. The other checks have a similar structure, and details can be taken from the proof.

- First, we want to make sure that A inputs the correct value c_A^l to obtain $M_{c_A^l}$. At the same time, B must not use a different Δ_A than for the other MACs. We compute a check value $\gamma^{(1)}$ as follows. A and B use another $\mathbb{F}_q$-OLE to compute $\sigma_A^{(1)} \leftarrow \mathbb{F}_q\text{-OLE}(-K_{a_A}, K_{c_A^l} + \gamma^{(1)}; b_A)$, where B chooses $\gamma^{(1)} \leftarrow_\$ \mathbb{F}_q$. Now A locally computes $b_A M_{a_A} + \sigma^{(1)} - M_{c_A^l}$, which evaluates to $\gamma^{(1)}$ if $M_{c_A^l}$ was computed with the correct c_A^l and Δ_A.
  Notice that the computation of $b_A M_{a_A}$ yields the important part of $M_{c_A^l}$, namely $\Delta_A a_A b_A$, plus some additional values which are "corrected" via $\sigma_A^{(1)}$. Since M_{a_A} uses the correct a_A, and the input b_A into the $\mathbb{F}_q$-OLE has to be consistent with M_{b_A} (this is ensured separately), the input to obtain $M_{c_A^l}$ has to be $a_A b_A$. By the same argumentation, if B uses a different Δ_A for $M_{c_A^l}$, due to the use of M_{a_A} the check value will be uniformly random.
- Secondly, we have to ensure that c_A^i is correctly computed. Another OLE is used to compute $\sigma^{(2)} \leftarrow \mathbb{F}_q\text{-OLE}(M_{a_B}, M_{r_B}; b_A)$. A locally computes $\sigma^{(2)\prime} = \Delta_B c_A^i + K_{a_B} b_A + K_{r_B}$, and if B used the correct inputs, it holds that $\sigma^{(2)\prime} = \sigma^{(2)}$.
- Now that we know that c_A^i is correct, we have to verify that the MAC $M_{c_A^i}$ was generated correctly. Towards this, we first observe that we can create $M_{c_A^i}$ in a different way than described above: $M_{c_A^i} \leftarrow \mathbb{F}_q\text{-OLE}(\Delta_A a_B, \Delta_A r_B + K_{c_A^i}; b_A)$ yields $M_{c_A^i} = \Delta_A(a_B b_A + r_B) + K_{c_A^i}$. Now we create another check value $\gamma^{(2)}$ by using an additional authenticated value s_A. A locally computes $d = c_A^i - s_A$ and $\gamma^{(2)} = M_{c_A^i} - M_{s_A}$, and sends d to B. B locally computes $\Delta_A d + K_{c_A^i} - K_{s_A}$. Again, if A used the correct input for the MAC generation, and B as well, both parties will obtain the same $\gamma^{(2)}$.
- It remains to bind the input of A to the before authenticated value b_A. We therefore add a final check $\gamma^{(3)}$: A obtains $\sigma_A^{(3)} \leftarrow \mathbb{F}_q\text{-OLE}(\Delta_A, \sigma_B^{(3)}; b_A)$. Now A locally computes $\gamma^{(3)} = M_{b_A} - \sigma_A^{(3)}$, while B locally computes $K_{b_A} - \sigma_B^{(3)}$.

In all of the above subprotocols, if a malicious party uses inconsistent inputs for the computation of the output and check values, the check values will depend on one of the other party's secret inputs (i.e. MAC key or share), and thus they will be uniformly distributed from the malicious party's view. In a final step we add all $\gamma^{(i)}$'s to global check values γ^{glo_i}. Then A commits to its set of γ^{glo_i}, B sends its set to A, and A has to unveil. If the values do not match, the protocol is aborted.

In the final protocol, we make use of the previously described $\mathbb{F}_q^k$-OLE primitive, which allows us to directly authenticate a batch of inputs, and also ensures that A has to use the same input b_A in all of the above mentioned steps. The complete description is given in Figure 8.

Π_{Deal}

Initialize: On input (`init`), A (resp. B) samples $\Delta_B \in \mathbb{F}_q$ (resp. Δ_A) uniformly at random and outputs the value. Both parties execute the following protocol twice in parallel, one where they play sender and one where they play receiver.

(1) S: On input (`aValue`, n), draw two random vectors $\mathbf{v}, \mathbf{r} \in \mathbb{F}_q^n$ and send $\mathbf{v}, \mathbf{r}$ to $\mathbb{F}_q^n$-OLE.
(2) R: Draw a random vector $\mathbf{K} \in \mathbb{F}_q^n$. Input Δ_S into $\mathbb{F}_q^n$-OLE and obtain $\Delta_S \mathbf{v} + \mathbf{r}$. Send $\mathbf{w} = \mathbf{v}\Delta_S + \mathbf{r} + \mathbf{K}$ to S. Store $\mathbf{K}$.
(3) S: Set $\mathbf{M} = \mathbf{w} - \mathbf{r}$ and store $(\mathbf{v}, \mathbf{M})$.

Authenticated Value(A): Pick an unused index i. A outputs v_i, M_{v_i} and B outputs K_{v_i}, i is marked as used.

Authenticated Value(B): Symmetrical to **AuthenticatedValue**(A)

Figure 7: Authentication step of protocol Π_{Deal} that realizes $\mathcal{F}_{\text{Deal}}$ in the $(\mathbb{F}_q$-OLE, $\mathbb{F}_q^k$-OLE$)$-hybrid model.

Theorem 4.1. *The protocol Π_{Deal} in Figures 7 and 8 UC-realizes $\mathcal{F}_{\text{Deal}}$ in the $(\mathbb{F}_q$-OLE, $\mathbb{F}_q^k$-OLE$)$-hybrid model with unconditional security and constant communication and computation overhead.*

Proof. Since the protocol Π_{Deal} is executed symmetrically by both parties, our proof strategy proceeds as follows. We provide two simulators against both corrupted sender and receiver in Π_{Deal}. We define a wrapper simulator which internally runs both simulators (since a party $P \in \{A, B\}$ plays both sender and receiver in a complete execution of Π_{Deal}). This wrapper simulator interacts with $\mathcal{F}_{\text{Deal}}$ and provides an indistinguishable simulation.

The wrapper simulator $\mathcal{S}_{\text{wrap}}$ (cf. Figure 9) first runs the initialization phase, where a complete state is created, i.e. all inputs of $\mathcal{A}_P$ are extracted, and a set of simulated inputs is generated. This state is then used by the internal simulators for the multiplication step in order to allow the extraction of $\mathcal{A}_P$'s secrets.

Π_{Deal} cont'd

Let COM be the commitment from Section 3.1.

Authenticated Multiplication: Both parties execute the following protocol twice in parallel, one where they play sender and one where they play receiver. We split the multiplication into several parts.

Inputs: S: Call **Authenticated Value(S)** four times to obtain $[a]_S$, $[b]_S$, $[r]_S$, $[s]_S$.

Multiplication of local shares: The result is $[c^l]_S = [a_S b_S]_S$.

(1) S: Compute $c_S^l = a_S \cdot b_S$. Input c_S^l into $\mathbb{F}_q$-OLE.

(2) R: Draw $K_{c_S^l} \in \mathbb{F}_q$ uniformly at random, input $(\Delta_S, K_{c_S^l})$ into $\mathbb{F}_q$-OLE and output $K_{c_S^l}$.

(3) S: Obtain $M_{c_S^l} = \Delta_S c_S^l + K_{c_S^l}$ and output $(c_S^l, M_{c_S^l})$.

Multiplication of cross shares: The result is $[c^i]_S = [a_R b_S + r_R]_S$.

(1) R: Draw $K_{c_S^i} \in \mathbb{F}_q$ uniformly at random and input $\big((a_R, \Delta_S a_R, -K_{a_S}, M_{a_R}, \Delta_S), (r_R, \Delta_S r_R + K_{c_S^i}, K_{c_S^l} + \gamma_R^{(1)}, M_{r_R}, \sigma_R^{(3)})\big)$ into $\mathbb{F}_q^5$-OLE.

(2) S: Input b_S into $\mathbb{F}_q^5$-OLE and obtain the following values:

- $c_S^i = a_R b_S + r_R$
- $M_{c_S^i} = \Delta_S(a_R b_S + r_R) + K_{c_S^i}$
- $\sigma_S^{(1)} = -b_S K_{a_S} + K_{c_S^l} + \gamma_R^{(1)}$
- $\sigma_S^{(2)} = M_{a_R} b_S + M_{r_R} = \Delta_R(r_R + b_S a_R) + K_{r_R} + K_{a_R} b_S$
- $\sigma_S^{(3)} = \Delta_S b_S + \sigma_R^{(3)}$

Output $(c_S^i, M_{c_S^i})$.

Checks and output: The result is $[a_S]_S$, $[b_S]_S$, $[c_S]_S$. Let $\gamma_P^{\mathsf{glo}_i}$, $i \in [3]$ for $P \in \{S, R\}$ be global checksums.

(1) Verification of $M_{c_S^l}$.

- S: Compute $\gamma_S^{(1)} = b_S M_{a_S} + \sigma_S^{(1)} - M_{c_S^l}$ and set $\gamma_S^{\mathsf{glo}_1}$ += $\gamma_S^{(1)}$.
- R: Set $\gamma_R^{\mathsf{glo}_1}$ += $\gamma_R^{(1)}$.

(2) Verification of c_S^i.

- S: Compute $\sigma_S^{(2)'} = \Delta_R c_S^i + K_{a_R} b_S + K_{r_R}$. Abort if $\sigma_S^{(2)'} \neq \sigma_S^{(2)}$. Further set $\gamma_S^{(3)} = M_{b_S} - \sigma_S^{(3)}$ and $\gamma_S^{\mathsf{glo}_3}$ += $\gamma_S^{(3)}$.
- R: Compute $\gamma_R^{(3)} = K_{b_S} - \sigma_R^{(3)}$ and set $\gamma_R^{\mathsf{glo}_3}$ += $\gamma_R^{(3)}$.

(3) Verification of $M_{c_S^i}$.

- S: Compute $\gamma_S^{(2)} = M_{c_S^i} - M_{s_S}$ and set $\gamma_S^{\mathsf{glo}_2}$ += $\gamma_S^{(2)}$. Send $d_S = c_S^i - s_S$ to R.
- R: Compute $\gamma_R^{(2)} = \Delta_S d_S + K_{c_S^i} - K_{s_S}$ and set $\gamma_R^{\mathsf{glo}_2}$ += $\gamma_R^{(2)}$.

(4) Final check over all n triples:

- S: Compute (com, unv) = COM.Commit($\gamma_S^{\mathsf{glo}_1}$, $\gamma_S^{\mathsf{glo}_2}$, $\gamma_S^{\mathsf{glo}_3}$) and send com to R.
- R: Send ($\gamma_R^{\mathsf{glo}_1}$, $\gamma_R^{\mathsf{glo}_2}$, $\gamma_R^{\mathsf{glo}_3}$) to S.
- S: If $\gamma_R^{\mathsf{glo}_i} \neq \gamma_S^{\mathsf{glo}_i}$ for any $i \in [3]$, abort. Otherwise send unv to R, set $[c_S]_S^{(j)} = [c^l]_S^{(j)} - [r]_S^{(j)} + [c^i]_S^{(j)}$ and output $(a_S^{(j)}, M_{a_S}^{(j)}, b_S^{(j)}, M_{b_S}^{(j)}, c_S^{(j)}, M_{c_S}^{(j)})$ for $j \in [n]$.
- R: If $\gamma_R^{\mathsf{glo}_i} \neq \gamma_S^{\mathsf{glo}_i}$ for any $i \in [3]$, abort, otherwise output $(K_{a_S}^{(j)}, K_{b_S}^{(j)}, K_{c_S}^{(j)})$ for $j \in [n]$.

Figure 8: Multiplication step of protocol Π_{Deal} that realizes $\mathcal{F}_{\mathsf{Deal}}$ in the ($\mathbb{F}_q$-OLE, $\mathbb{F}_q^k$-OLE)-hybrid model.

In the following, we will show that for all environments $\mathcal{Z}$, it holds that

$$\mathsf{Real}^{\mathcal{A}_P}_{\Pi_{\mathsf{Deal}}} \approx_s \mathsf{Ideal}^{\mathcal{S}_{\mathsf{wrap}}(P)}_{\mathcal{F}_{\mathsf{Deal}}}.$$

Simulator $\mathcal{S}_{\mathsf{wrap}}(P)$

Initialize: Run $\mathcal{S}_S.\mathcal{S}_{\mathsf{ini}}$ with S = P to obtain $(\mathbf{v}_P, M_{\mathbf{v}_P}, \hat{\Delta}_P, \hat{\mathbf{K}}_{\mathbf{v}_P})$ and $\mathcal{S}_R.\mathcal{S}_{\mathsf{ini}}$ with R = P to obtain $(\Delta_{\bar{P}}, K_{\mathbf{v}_{\bar{P}}}, \hat{\mathbf{v}}_{\bar{P}}, \hat{\mathbf{M}}_{\mathbf{v}_{\bar{P}}})$. Store state = $((\Delta_{\bar{P}}, \mathbf{K}_{\mathbf{v}_{\bar{P}}}, \mathbf{v}_P, \mathbf{M}_{\mathbf{v}_P}), (\hat{\Delta}_P, \hat{\mathbf{K}}_{\mathbf{v}_P}, \hat{\mathbf{v}}_{\bar{P}}, \hat{\mathbf{M}}_{\mathbf{v}_{\bar{P}}}))$ and send (setInit, P, $(\Delta_{\bar{P}}, \mathbf{v}_P, M_{\mathbf{v}_P}, K_{\mathbf{v}_{\bar{P}}})$) to $\mathcal{F}_{\mathsf{Deal}}$.

Authenticated value: Simulate according to Π_{Deal}.

Authenticated multiplication: Run $\mathcal{S}_S.\mathcal{S}_{\mathsf{mul}}(\mathsf{state})$ with S = P to obtain (c_P, M_{c_P}) and $\mathcal{S}_R.\mathcal{S}_{\mathsf{mul}}(\mathsf{state})$ with R = P to obtain $K_{c_{\bar{P}}}$. Send (setMult, P, $(c_P, M_{c_P}, K_{c_{\bar{P}}})$) to $\mathcal{F}_{\mathsf{Deal}}$.

Figure 9: Simulator $\mathcal{S}_{\mathsf{wrap}}$ against a corrupted party $P \in \{A, B\}$.

Towards this, we define two helper simulators $\mathcal{S}_S$ and $\mathcal{S}_R$, and show that their simulation of the other party $\bar{P}$ is indistinguishable from a real protocol run.

Corrupted sender. The simulator $\mathcal{S}_S$ (cf. Figure 10) creates a complete set of keys for the initialization phase and extracts the corresponding MACs and inputs of $\mathcal{A}_S$. For the multiplication phase, it is given as input a complete state of simulated and extracted inputs, keys and MACs. Based on these, the simulator can compare $\mathcal{A}_S$'s input with the correct input and abort if they do not match.

Simulator $\mathcal{S}_S$

$\mathcal{S}_S$ formally consists of two algorithms $\mathcal{S}_{\mathsf{ini}}$, $\mathcal{S}_{\mathsf{mul}}$, where $\mathcal{S}_{\mathsf{mul}}$ gets an input $((\Delta_R, \mathbf{K}_{\mathbf{v}R}, \mathbf{v}_S, \mathbf{M}_{\mathbf{v}S}), (\hat{\Delta}_S, \hat{\mathbf{K}}_{\mathbf{v}S}, \hat{\mathbf{v}}_R, \hat{\mathbf{M}}_{\mathbf{v}R}))$.

Initialize ($\mathcal{S}_{\mathsf{ini}}$): Sample $\hat{\Delta}_S \leftarrow_\$ \mathbb{F}_q$. Extract $\mathcal{A}_S$'s inputs into $\mathbb{F}_q^{4n}$-OLE and obtain $\mathbf{v}_S$ and $\mathbf{r}_S$. Draw $\hat{\mathbf{K}}_{\mathbf{v}_S} \leftarrow_\$ \mathbb{F}_q^{4n}$, compute $\mathbf{M}_{\mathbf{v}_S} = \Delta_S \mathbf{v}_S + \mathbf{K}_{\mathbf{v}_S}$ and $\mathbf{w} = \mathbf{M}_{\mathbf{v}_S} + \mathbf{r}_S$. Send $\mathbf{w}$ to $\mathcal{A}_S$ and output $(\mathbf{v}_S, \mathbf{M}_{\mathbf{v}_S}, \hat{\Delta}_S, \hat{\mathbf{K}}_{\mathbf{v}_S})$.

Authenticated value: Simulate according to Π_{Deal}.

Authenticated multiplication ($\mathcal{S}_{\mathsf{mul}}$): Simulate Π_{Deal} and let ξ_1 be $\mathcal{A}_S$'s input into $\mathbb{F}_q$-OLE and ξ_2 be the input into $\mathbb{F}_q^5$-OLE. Further let ξ_3 be the message from $\mathcal{A}_S$.

- If $\xi_1 \neq a_S b_S$ and $\xi_2 \neq b_S$, set $\gamma_R^{(1)} \leftarrow_\$ \mathbb{F}_q$.
- If $\xi_2 \neq b_S$, set $\gamma_R^{(3)} \leftarrow_\$ \mathbb{F}_q$.
- If $\xi_3 \neq c_S^i - s_S$, set $\gamma_R^{(2)} \leftarrow_\$ \mathbb{F}_q$.

Continue the simulation according to Π_{Deal} and compute $c_S = c_S^l + c_S^i - r_S$ and $M_{c_S} = M_{c_S^l} + M_{c_S^i} - M_{r_S}$ if no abort occurs. Output (c_S, M_{c_S}).

Figure 10: Simulator $\mathcal{S}_S$ against a corrupted sender.

Let Δ_R, $(\mathbf{K}_{\mathbf{a}_R}, \mathbf{K}_{\mathbf{b}_R}, \mathbf{K}_{\mathbf{r}_R}, \mathbf{K}_{\mathbf{s}_R})$ be $\mathcal{A}_S$'s input to the first $\mathbb{F}_q^{4n}$-OLE and $(\mathbf{a}_S, \mathbf{b}_S, \mathbf{r}_S, \mathbf{s}_S)$ the input and $(\mathbf{M}_{\mathbf{a}_S}, \mathbf{M}_{\mathbf{b}_S}, \mathbf{M}_{\mathbf{r}_S}, \mathbf{M}_{\mathbf{s}_S})$ the output of the second $\mathbb{F}_q^{4n}$-OLE. Further let ξ_1 be the input into $\mathbb{F}_q$-OLE, ξ_2 be the input into $\mathbb{F}_q^5$-OLE and ξ_3 be the message sent for the verification of $M_{c_S^i}$ (d_S in the honest case).

Consider the following hybrids $\mathcal{H}_0, \ldots, \mathcal{H}_n$.

$\mathcal{H}_0$**:** The real experiment.

$\mathcal{H}_{1,\ldots,n}$ For $i = 1, \ldots, n$, $\mathcal{H}_i$ is identical to $\mathcal{H}_{i-1}$, except for the following. In round i, let ξ_1 be $\mathcal{A}_S$'s input into the $\mathbb{F}_q$-OLE and let ξ_2 be the input into $\mathbb{F}_q^5$-OLE. Further let ξ_3 be the message sent by $\mathcal{A}_S$ to R. If $(\xi_1, \xi_2, \xi_3) \neq (a_S b_S, b_S, d_S)$, set $\gamma_R^{\mathsf{glo}_i} \leftarrow_\$ \mathbb{F}_q$ for the corresponding incorrect ξ_i.

$\mathcal{H}_n$: The ideal experiment.

We first observe that for each round i, every relevant value of the receiver unknown to $\mathcal{A}_S$ can be expressed in terms of Δ_S, i.e. we have that

$$r_R = c_S^i - a_R\xi_2 \tag{5}$$
$$K_{a_S} = M_{a_S} - \Delta_S a_S \tag{6}$$
$$K_{b_S} = M_{b_S} - \Delta_S b_S \tag{7}$$
$$K_{s_S} = M_{s_S} - \Delta_S s_S \tag{8}$$
$$K_{c_S^l} = M_{c_S^l} - \Delta_S\xi_1 \tag{9}$$
$$K_{c_S^i} = M_{c_S^i} - \Delta_S(a_R\xi_2 + r_R) = M_{c_S^i} - \Delta_S c_S^i \tag{10}$$
$$\begin{aligned}\gamma_R^{(1)} &= \sigma_S^{(1)} + K_{a_S}\xi_2 - K_{c_S^l}\\ &= \sigma_S^{(1)} + (M_{a_S} - \Delta_S a_S)\xi_2 - (M_{c_S^l} - \Delta_S\xi_1)\\ &= \sigma_S^{(1)} + \Delta_S(\xi_1 - a_S\xi_2) + \xi_2 M_{a_S} - M_{c_S^l}\end{aligned} \tag{11}$$
$$\sigma_R^{(3)} = \sigma_S^{(3)} - \Delta_S\xi_2 \tag{12}$$

Thus, all of these values are uniformly distributed given $\mathcal{A}_S$'s view.

We will now show that for $i = 1, \ldots, n$, the hybrids $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ are statistically close. Clearly, if $(\xi_1, \xi_2, \xi_3) = (a_S b_S, b_S, d_S)$, then the two experiments are identical given the view of $\mathcal{Z}$. Therefore, the only way to distinguish $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ is to provide inputs $(\xi_1, \xi_2, \xi_3) \neq (a_S b_S, b_S, d_S)$ and pass the checks of γ^{glo_i}.

The abort condition for the check $\gamma^{(1)}$ can be rewritten as

$$\begin{aligned}0 &= b_S M_{a_S} + \sigma_S^{(1)} - M_{c_S^l} - \gamma_R^{(1)}\\ &= b_S M_{a_S} + \sigma_S^{(1)} - M_{c_S^l} - (\sigma_S^{(1)} + \Delta_S(\xi_1 - a_S\xi_2) + \xi_2 M_{a_S} - M_{c_S^l})\\ &= \Delta_S(a_S\xi_2 - \xi_1) + M_{a_S}(b_S - \xi_2)\end{aligned} \tag{13}$$

using Equation (11). By applying Equations (8) and (10) we can rewrite the check of $\gamma^{(2)}$ as

$$\begin{aligned}0 &= M_{c_S^i} - M_{s_S} - \Delta_S\xi_3 - K_{c_S^i} + K_{s_S}\\ &= M_{c_S^i} - M_{s_S} - \Delta_S\xi_3 - M_{c_S^i} + \Delta_S c_S^i + M_{s_S} - \Delta_S s_S\\ &= \Delta_S(c_S^i - s_S - \xi_3)\end{aligned} \tag{14}$$

Finally, for the check $\gamma^{(3)}$ we get

$$\begin{aligned}0 &= M_{b_S} - \sigma_S^{(3)} - K_{b_S} + \sigma_R^{(3)}\\ &= M_{b_S} - \sigma_S^{(3)} - M_{b_S} + \Delta_S b_S + \sigma_S^{(3)} - \Delta_S\xi_2\\ &= \Delta_S(b_S - \xi_2).\end{aligned} \tag{15}$$

by applying Equations (7) and (12).

Clearly, if $\xi_1 = a_S b_S$ and $\xi_2 = b_S$, Equations (13) and (15) hold. But if $\xi_1 \neq a_S b_S$, then $\gamma_R^{(1)}$ is uniformly distributed given $\mathcal{A}_S$'s view and the check of $\gamma^{(1)}$ will fail. On the other hand, if $\xi_2 \neq b_S$, then the check for $\gamma^{(3)}$ fails.

Similarly, if $\xi_3 \neq c_S^i - s_S$, $\gamma_R^{(2)}$ will be uniformly distributed given $\mathcal{A}_S$'s view. Thus, if $(\xi_1, \xi_2, \xi_3) \neq (a_S b_S, b_S, c_S^i - s_S)$, the probability that $\mathcal{A}_S$ passes the check for γ^{glo_i} for $i \in [3]$ is upper bounded by $3/|\mathbb{F}_q|$. This yields that given $\mathcal{Z}$'s view, the statistical distance between the hybrids $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ is at most $1/|\mathbb{F}_q|$. Combined, the statistical distance between $\mathcal{H}_0$ and $\mathcal{H}_n$ is bounded by $3n/|\mathbb{F}_q|$, which is negligible in the security parameter. This establishes a correct simulation against a corrupted sender in Π_{Deal}.

Corrupted receiver. The simulator $\mathcal{S}_R$ against a corrupted receiver $\mathcal{A}_R$ is conceptually very simple. During the initialization, it simply extracts the global MAC key and all other keys from $\mathcal{A}_R$ and also creates a consistent set of inputs and MACs from these keys. For the multiplication, it is given a complete set of all inputs, MACs and keys of the $\mathcal{A}_R$. This allows the simulator to compare the actual inputs during the multiplication protocol with the intended inputs. If they do not match, the simulator simply sets the check values to a random value, which forces the protocol to fail. The simulator is described in Figure 11.

Simulator $\mathcal{S}_R$

$\mathcal{S}_R$ formally consists of two algorithms $\mathcal{S}_{\mathsf{ini}}$, $\mathcal{S}_{\mathsf{mul}}$, where $\mathcal{S}_{\mathsf{mul}}$ gets an input $((\Delta_S, \mathbf{K}_{\mathbf{v}S}, \mathbf{v}_R, \mathbf{M}_{\mathbf{v}R}), (\hat{\Delta}_R, \hat{\mathbf{K}}_{\mathbf{v}_R}, \hat{\mathbf{v}}_S, \hat{\mathbf{M}}_{\mathbf{v}_S}))$.

Initialize ($\mathcal{S}_{\mathsf{ini}}$): Sample $\hat{\mathbf{v}}_S, \mathbf{r}_S \leftarrow_\$ \mathbb{F}_q^{4n}$. Extract $\mathcal{A}_R$'s input into $\mathbb{F}_q^{4n}$-OLE and obtain Δ_S. After obtaining $\mathbf{w}$, compute $\hat{\mathbf{M}}_{\mathbf{v}_S} = \mathbf{w} - \mathbf{r}_S$ and $\mathbf{K}_{\mathbf{v}_S} = \mathbf{M}_{\mathbf{v}_S} - \Delta_S\mathbf{v}_S$. Output $(\Delta_S, \mathbf{K}_{\mathbf{v}_S}, \hat{\mathbf{v}}_S, \hat{\mathbf{M}}_{\mathbf{v}_S})$.

Authenticated value: Simulate according to Π_{Deal}.

Authenticated multiplication ($\mathcal{S}_{\mathsf{mul}}$): Simulate Π_{Deal} and let (α_1, β_1) be $\mathcal{A}_R$'s input into $\mathbb{F}_q$-OLE and $\big((\alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6), (\beta_2, \beta_3, \beta_4, \beta_5, \beta_6)\big)$ be the inputs into $\mathbb{F}_q^5$-OLE.

- If $(\alpha_1, \alpha_4) \neq (\Delta_S, -K_{a_S})$, set $\gamma_S^{(1)} \leftarrow_\$ \mathbb{F}_q$.
- If $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_R, M_{a_R}, r_R, M_{r_R})$, abort.
- If $(\alpha_2, \alpha_3) \neq (a_R, \Delta_S a_R)$, set $\gamma_S^{(2)} \leftarrow_\$ \mathbb{F}_q$.
- If $\alpha_6 \neq \Delta_S$, set $\gamma_S^{(3)} \leftarrow_\$ \mathbb{F}_q$.

Continue the simulation according to Π_{Deal} and compute $K_{c_S} = K_{c_S^l} + K_{c_S^i} - K_{r_S}$, if no abort occurs. Output K_{c_S}.

Figure 11: Simulator $\mathcal{S}_R$ against a corrupted receiver.

Let Δ_S, $(\mathbf{K}_{\mathbf{a}_S}, \mathbf{K}_{\mathbf{b}_S}, \mathbf{K}_{\mathbf{r}_S}, \mathbf{K}_{\mathbf{s}_S})$ be $\mathcal{A}_R$'s input to the first $\mathbb{F}_q^{4n}$-OLE and $(\mathbf{a}_R, \mathbf{b}_R, \mathbf{r}_R, \mathbf{s}_R)$ the input and $(\mathbf{M}_{\mathbf{a}_R}, \mathbf{M}_{\mathbf{b}_R}, \mathbf{M}_{\mathbf{r}_R}, \mathbf{M}_{\mathbf{s}_R})$ the output of the second $\mathbb{F}_q^{4n}$-OLE. Further let (α_1, β_1) be $\mathcal{A}_R$'s input to $\mathbb{F}_q$-OLE and $(\alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6), (\beta_2, \beta_3, \beta_4, \beta_5, \beta_6)$ be the inputs to $\mathbb{F}_q^5$-OLE.

Consider the following hybrids $\mathcal{H}_0, \ldots, \mathcal{H}_{2n}$.

$\mathcal{H}_0$: The real experiment.

$\mathcal{H}_{\{1,\ldots,n\}}$: For $i = 1, \ldots, n$, $\mathcal{H}_i$ is identical to $\mathcal{H}_{i-1}$, except for the following. If $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_R, M_{a_R}, r_R, M_{a_R})$ in round i, abort regardless of whether $\sigma_S^{(2)} = \Delta_R c_S^i + K_{a_R} b_S + K_{r_R}$.

$\mathcal{H}_{\{n+1,\ldots,2n\}}$: For $i = n+1, \ldots, 2n$, $\mathcal{H}_i$ is identical to $\mathcal{H}_{i-1}$, except for the following. In the case that $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_6) \neq (\Delta_S, a_R, \Delta_S a_R, -K_{a_S}, \Delta_S)$, set $\gamma_S^{\mathsf{glo}_i} \leftarrow_\$ \mathbb{F}_q$ for the corresponding α_i.

$\mathcal{H}_{2n}$: The ideal experiment.

We first observe that for each round i, every value of the sender unknown to $\mathcal{A}_R$ can be expressed in terms of the MAC key Δ_R and

inputs a_S, b_S and s_S, i.e. we have that

$$K_{a_R} = M_{a_R} - \Delta_R a_R \tag{16}$$
$$K_{r_R} = M_{r_R} - \Delta_R r_R \tag{17}$$
$$M_{a_S} = \Delta_S a_S + K_{a_S} \tag{18}$$
$$M_{b_S} = \Delta_S b_S + K_{b_S} \tag{19}$$
$$M_{s_S} = \Delta_S s_S + K_{s_S} \tag{20}$$
$$M_{c_S^l} = \alpha_1 a_S b_S + \beta_1 \tag{21}$$
$$M_{c_S^i} = \alpha_3 b_S + \beta_3 \tag{22}$$
$$c_S^i = \alpha_2 b_S + \beta_2 \tag{23}$$
$$d_S = c_S^i - s_S = \alpha_2 b_S + \beta_2 - s_S \tag{24}$$
$$\sigma_S^{(1)} = \alpha_4 b_S + \beta_4 \tag{25}$$
$$\sigma_S^{(2)} = \alpha_5 b_S + \beta_5 \tag{26}$$
$$\sigma_S^{(3)} = \alpha_6 b_S + \beta_6 \tag{27}$$

Thus, all of these values are uniformly distributed given $\mathcal{A}_S$'s view. We will start by showing that for $i = 1, \ldots, n$ the hybrids $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ are statistically close to the view of $\mathcal{Z}$. $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ differ only in round i. If it holds that $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_R, M_{a_R}, r_R, M_{a_R})$, then the sender will always abort in $\mathcal{H}_i$, possibly allowing $\mathcal{Z}$ to distinguish. We will now show that in case $(\alpha_1, \alpha_2, \beta_1, \beta_2) \neq (a_R, M_{a_R}, r_R, M_{a_R})$ the sender also aborts with overwhelming probability in $\mathcal{H}_{i-1}$, establishing that the two hybrids are statistically close.

We rewrite the abort condition of $\sigma^{(2)}$ as follows using Equations (16), (17), (23) and (26).

$$\begin{aligned}
0 &= \Delta_R c_S^i + K_{a_R} b_S + K_{r_R} - \sigma_S^{(2)} \\
&= \Delta_R(\alpha_2 b_S + \beta_2) + K_{a_R} b_S + K_{r_R} - \alpha_5 b_S - \beta_5 \\
&= \Delta_R \alpha_2 b_S + \Delta_R \beta_2 + (M_{a_R} - \Delta_R a_R) b_S + M_{r_R} - \Delta_R r_R - \alpha_5 b_S - \beta_5 \\
&= b_S(M_{a_R} - \alpha_5) + \beta_5 - M_{r_R} + \Delta_R((\alpha_2 - a_R) b_S + \beta_2 - r_R)
\end{aligned} \tag{28}$$

Define the random variables

$$L = b_S(\alpha_5 - M_{a_R}) + \beta_5 - M_{r_R}$$
$$R = b_S(\alpha_2 - a_R) + \beta_2 - r_R,$$

and we can simplify Equation (28) into

$$L = \Delta_R \cdot R.$$

Assume first that $\alpha_2 = a_R$ and $\beta_2 = r_R$. In this case the random variable R is constant 0 and it must either hold that $\alpha_5 \neq M_{a_R}$ or $\beta_5 \neq M_{r_R}$. Therefore, the variable L is either constant non-zero or uniformly random over $\mathbb{F}_q$. Thus, the probability that Equation (28) holds is at most $1/|\mathbb{F}_q|$.

Now, assume that $\alpha_2 \neq a_R$ or $\beta_2 \neq r_R$. In this case the random variable R is either constant non-zero or uniformly random over $\mathbb{F}_q$. If R is non-zero, then the probability that Equation (28) holds is at most $1/|\mathbb{F}_q|$ over the choice of Δ_R. Thus

$$\begin{aligned}
\Pr[L = \Delta_R \cdot R] &= \underbrace{\Pr[R \neq 0]}_{<1} \cdot \underbrace{\Pr[L = \Delta_R \cdot R | R \neq 0]}_{\leq 1/|\mathbb{F}_q|} \\
&\quad + \underbrace{\Pr[R = 0]}_{\leq 1/|\mathbb{F}_q|} \cdot \underbrace{\Pr[L = \Delta_R \cdot R | R = 0]}_{\leq 1} \\
&\leq 2/|\mathbb{F}_q|.
\end{aligned}$$

We conclude that if $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_R, M_{a_R}, r_R, M_{a_R})$, then the abort condition is triggered, except with probability at most $2/|\mathbb{F}_q|$. Thus, the statistical distance between $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ is at most $2/|\mathbb{F}_q|$.

We will now show that for $i = n+1, \ldots, 2n$, the hybrids $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ are statistically close. Clearly, if the inputs are according to Π_{Deal}, then the two experiments are identical given the view of $\mathcal{Z}$. Therefore, the only way to distinguish $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ is to provide inputs $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_6) \neq (\Delta_S, a_R, \Delta_S a_R, -K_{a_S}, \Delta_S)$ and pass the check of the γ^{glo_i}'s.

Towards this, we rewrite all checks for $\gamma^{(1)}, \gamma^{(2)}$ and $\gamma^{(3)}$ using the above identities. For $\gamma^{(1)}$, we apply Equations (18), (21) and (25) and get

$$\begin{aligned}
0 &= b_S M_{a_S} + \sigma_S^{(1)} - M_{c_S^l} \\
&= b_S(\Delta_S a_S + K_{a_S}) + \alpha_4 b_S + \beta_4 - (\alpha_1 a_S b_S + \beta_1) \\
&= b_S((\Delta_S - \alpha_1) a_S + K_{a_S} + \alpha_4) + \beta_4 - \beta_1
\end{aligned} \tag{29}$$

If $\alpha_1 = \Delta_S$ but $\alpha_4 \neq -K_{a_S}$, the term depending on b_S in Equation (29) does not vanish, hence the result will be uniformly distributed. The same holds for the case $\alpha_1 \neq \Delta_S$ because of the dependence on the unknown a_S. Combined, if $(\alpha_1, \alpha_4) \neq (\Delta_S, -K_{a_S})$, the value $\gamma^{(1)}$ is uniformly distributed given $\mathcal{A}_R$'s view.

Moving on, the check for $\gamma^{(2)}$ can be rewritten as

$$\begin{aligned}
0 &= M_{c_S^i} - M_{s_S} - (\Delta_S d_S + K_{c_S^i} - K_{s_S}) \\
&= \alpha_3 b_S + \beta_3 - (\Delta_S s_S + K_{s_S}) - (\Delta_S(\alpha_2 b_S + \beta_2 - s_S) + K_{c_S^i} - K_{s_S}) \\
&= b_S(\alpha_3 - \Delta_S \alpha_2) + \beta_3 - \Delta_S \beta_2 - K_{c_S^i}
\end{aligned} \tag{30}$$

using the identities in Equations (20), (22) and (24). If $(\alpha_2, \alpha_3) \neq (a_R, M_{a_R})$, Equation (30) will only hold with probability $2/|\mathbb{F}_q|$. This follows from Equation (28) (i.e. it has to hold that $\alpha_2 = a_R$), and the fact that Equation (30) is uniformly distributed if $\alpha_3 \neq \Delta_S \alpha_2$.

Finally, we apply Equations (19) and (27) to the check for $\gamma^{(3)}$ and get

$$\begin{aligned}
0 &= M_{b_S} - \sigma_S^{(3)} - (K_{b_S} - \sigma_R^{(3)}) \\
&= \Delta_S b_S + K_{b_S} - \alpha_6 b_S - \beta_6 - K_{b_S} + \sigma_R^{(3)} \\
&= b_S(\Delta_S - \alpha_6) + \sigma_R^{(3)} - \beta_6
\end{aligned} \tag{31}$$

Thus, if $\alpha_6 \neq \Delta_S$, Equation (31) will be uniformly distributed. Combining all these observations, it follows that the probability that $\mathcal{A}_R$ produces a set of correct $\gamma_R^{\text{glo}_i}$ is upper bounded by $3/|\mathbb{F}_q|$, if $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_R, M_{a_R}, r_R, M_{a_R})$. Thus we can bound the statistical distance between $\mathcal{H}_i$ and $\mathcal{H}_{i-1}$ by $3/|\mathbb{F}_q|$.

We conclude that the statistical distance between $\mathcal{H}_0$ and $\mathcal{H}_{2n}$ can be upper bounded by $5n/|\mathbb{F}_q|$, which is negligible in the security parameter. □ □

Efficiency. For a complete authenticated triple, Π_{Deal} requires (amortized) 22 calls to $\mathbb{F}_q$-OLE:

- 6 for $\mathbb{F}_q^5$-OLE; the check requiring the commitment in the $\mathbb{F}_q^k$-OLE protocol can be summed over all triples (it is a random value) and is thus amortized away.
- 4 for the MACs on the inputs; again both the additional $\mathbb{F}_q$-OLE and the commitment for $\mathbb{F}_q^k$-OLE are amortized over all triples.
- One for the MAC on the locally computed share.
- The verification of the γ^{glo_i} is only done once over all triples and thus amortized away.

We point out that even in the semi-honest setting, 8 calls to $\mathbb{F}_q$-OLE would be necessary to generate such a triple.

Regarding computational efficiency, note that the protocol only requires a constant number of basic field operations per triple.

5 SECURE TWO-PARTY COMPUTATION

In this section we describe the two-party computation protocol for arithmetic circuits. We prove its UC-security in the $\mathcal{F}_{\text{Deal}}$-hybrid model. The protocol construction uses basic techniques to turn randomized arithmetic operations into deterministic arithmetic operations. Since we use authenticated values as input, we have to make sure that an arithmetic operation returns an authenticated result. All operations needed to de-randomize the inputs are additions. Suppose party A wants to add two authenticated values (x, M_x) and (y, M_y) with corresponding keys K_x and K_y held by B. Then A computes $z = x + y, M_z = M_x + M_y = \Delta_A(x + y) + K_x + K_y$, and B computes $K_z = K_x + K_y$. Then the value z is properly authenticated by M_z, and B holds the corresponding key K_z.

$\mathcal{F}_{\text{2PC}}$

Rand: On input (rand, vid) from A and B, with vid being a fresh identifier, sample $r \in \mathbb{F}_q$ and store (vid, r)

Input: On input (inp, P, vid, x) from P $\in$ {A, B} and (inp, P, vid) from $\bar{\text{P}}$, with vid a fresh identifier, store (vid, x).

Add: On input (add, vid_1, vid_2, vid_3) from both parties, with vid_3 being a fresh identifier, retrieve (vid_1, x), (vid_2, y) and store (vid_3, $x + y$).

Multiply: On input (mult, vid_1, vid_2, vid_3) from both parties, with vid_3 being a fresh identifier, retrieve (vid_1, x), (vid_2, y) and store (vid_3, $x \cdot y$).

Output: On input (output, P, vid) from both parties, retrieve (vid, x) and output it to P.

Figure 12: Ideal arithmetic two-party computation functionality.

Theorem 5.1. *The protocol Π_{2PC} in Figure 13 UC-realizes $\mathcal{F}_{\text{2PC}}$ in the $\mathcal{F}_{\text{Deal}}$-hybrid model.*

Proof. **Correctness.** Correctness of the protocol is immediate with the exception of the multiplication step. Let $[a] = [a_A|a_B]$,$[b] = [b_A|b_B]$ and $[c] = [c_A|c_B]$ such that $c = a \cdot b$. Further let $\delta_{a_A} = a_A - x_A$, $\delta_{b_A} = b_A - y_A$, $\delta_{a_B} = a_B - x_B$, $\delta_{b_B} = b_B - y_B$, $\delta_x = \delta_{a_A} + \delta_{a_B}$ and $\delta_y = \delta_{b_A} + \delta_{b_B}$. Then

$$\begin{aligned}(a_A + a_B) \cdot (b_A + b_B) &= (x_A + \delta_{a_A} + x_B + \delta_{a_B}) \cdot (y_A + \delta_{b_A} + y_B + \delta_{b_B}) \\ &= (x_A + x_B + \delta_x) \cdot (y_A + y_B + \delta_y) \\ &= x \cdot y + y_A\delta_x + y_B\delta_x + x_A\delta_y + x_B\delta_y + \delta_x\delta_y\end{aligned}$$

We thus can compute

$$\begin{aligned}z_A &= c_A - y_A\delta_x - x_A\delta_y - \delta_x\delta_y \\ z_B &= c_B - y_B\delta_x - x_B\delta_y\end{aligned}$$

The value $\delta_x\delta_y$ has to be authenticated via a random authenticated value r_A: A sends $r' = \delta_x\delta_y - r_A$ and $u = M_{r_A} + t$ to B, where $t \leftarrow_\$ \mathbb{F}_q$. B locally computes $M'_{\delta_x\delta_y} = \Delta_A r' + u$ and sends it to A, who obtains $M_{\delta_x\delta_y} = M'_{\delta_x\delta_y} - t$.

UC-security. Consider the simulator $\mathcal{S}$ in Figure 14. It is easy to verify that $\mathcal{S}$ is able to extract the inputs of corrupted parties and provide a correct output, as long as the adversary follows the protocol. The only way to actively deviate from the protocol is by sending inconsistent values, for which the simulator aborts. We now show that the simulator aborts only with negligible probability.

We prove through a series of hybrid experiments that the probability that the adversary $\mathcal{A}$ manages to forge a MAC value is equivalent to guessing the global key Δ. Since every MAC consists of a new random key K, the only way an adversary will gain an advantage over guessing a new MAC is by adding/subtracting existing MACs. The proof proceeds along the lines of [22].

Experiment 1 Sample a random global key $\Delta \in \mathbb{F}_q$. $\mathcal{A}_1$ is allowed to send queries (mac, v, l), where $v \in \mathbb{F}_q$ and l is a fresh label. For each such query, sample a new local key $K \in$

Protocol Π_{2PC}

Init: A receives Δ_B from $\mathcal{F}_{\text{Deal}}$, B receives Δ_A.

Rand: A and B ask $\mathcal{F}_{\text{Deal}}$ for random authenticated values $[r_A]_A$,$[r_B]_B$ and store $[r] = [r_A|r_B]$.

Input: If P = A, then A asks $\mathcal{F}_{\text{Deal}}$ for a random authenticated value $[x_A]_A$ and announces x_B = x - x_A. The parties build $[x_B]_B$ and define $[x] = [x_A|x_B]$ (cf. Figure 1). This step is performed symmetrically for B.

Add: A and B retrieve $[x]$, $[y]$ and compute $[z] = [x] + [y]$. For that, A computes $[z_A]_A = [x_A]_A + [y_A]_A$, symmetrically for B. This can be done locally.

Mult: A and B retrieve $[x]$, $[y]$ and ask $\mathcal{F}_{\text{Deal}}$ for a random authenticated multiplication $[a] = [a_A|a_B]$,$[b] = [b_A|b_B]$ and $[c] = [c_A|c_B]$ such that $c = a \cdot b$. Additionally, A retrieves a random authenticated value $[r_A]_A$ from $\mathcal{F}_{\text{Deal}}$. A and B compute $z = x \cdot y$ as follows:

(1) A computes $\delta_{a_A} = a_A - x_A$ and $\delta_{b_A} = b_A - y_A$. Symmetrically, B computes $\delta_{a_B} = a_B - x_B$ and $\delta_{b_B} = b_B - y_B$.
(2) A and B exchange all δ-values and compute $\delta_x = \delta_{a_A} + \delta_{a_B}$ and $\delta_y = \delta_{b_A} + \delta_{b_B}$.
(3) A computes $\delta = \delta_x \cdot \delta_y$. A and B create $[\delta_A]_A$ via $[r_A]_A$.
(4) A locally computes: $[z_A]_A = [c_A]_A - [x_A]_A\delta_y - [y_A]_A\delta_x - [\delta_A]_A$.
(5) B locally computes: $[z_B]_B = [c_B]_B - [x_B]_B\delta_y - [y_B]_B\delta_x$.

Thus we have $[z] = [z_A|z_B]$.

Output: The parties retrieve $[x] = [x_A|x_B]$. If A is to learn x, then B reveals x_B. If B is to learn x, then A reveals x_A.

Figure 13: Secure two-party computation in the $\mathcal{F}_{\text{Deal}}$-hybrid model.

Simulator $\mathcal{S}$

Simulate $\mathcal{F}_{\text{Deal}}$ to the parties and learn all shares, keys and MACs of the parties.

Rand Run the protocol honestly, send (rand, vid) to $\mathcal{F}_{\text{2PC}}$.

Input For the corrupted party C, compute $x = x_{\text{C}} + x_{\text{H}}$, where x_{C} was sent to C by $\mathcal{F}_{\text{Deal}}$ and x_{H} was sent to the honest party H. Send (inp, C, x) to $\mathcal{F}_{\text{2PC}}$.
For the honest party, use (inp, H, 0) as the dummy input to the simulated protocol.

Add Run the protocol honestly and call add on $\mathcal{F}_{\text{2PC}}$.

Multiply Run the protocol honestly. Call mult on $\mathcal{F}_{\text{2PC}}$.

Output If P = C, help open a value $[x]$ after obtaining x' from $\mathcal{F}_{\text{2PC}}$ and computing x_{C} from the outputs of $\mathcal{F}_{\text{Deal}}$. Compute $x_{\text{H}} = x_{\text{C}} + x'$ and $M_{x_{\text{H}}} = K_{x_{\text{H}}} + x_{\text{H}}\delta_{\text{H}}$ ($K_{x_{\text{H}}}$ and Δ_{H} are known from $\mathcal{F}_{\text{Deal}}$).

Abort if any of the values are inconsistent, i.e., a MAC check passes, although the message was not derived according to the outputs of $\mathcal{F}_{\text{Deal}}$.

Figure 14: Simulator for Π_{2PC}.

$\mathbb{F}_q$, store (l, K, v) and return $M = \Delta v + K$. If $\mathcal{A}_1$ outputs a query $(\text{break}, a_1, l_1, \ldots, a_p, l_p, v', M')$, such that (l_1, K_1, v_1) to (l_p, K_p, v_p) are stored, $a_i \in \{0, 1\}$ and no break-query has been sent, compute $K = \sum_{i=0}^{p} a_i K_i$ and $v = \sum_{i=0}^{p} a_i v_i$. If $v' \neq v$ and $M' = \Delta v' + K$ $\mathcal{A}_1$ wins.

Experiment 2 Sample a random global key $\Delta \in \mathbb{F}_q$. $\mathcal{A}_2$ is allowed to send queries (mac, v, l, M), where $v, M \in \mathbb{F}_q$ and l is a fresh label. For each such query, store (l, K, v) and return $K = M - v\Delta$. If $\mathcal{A}_2$ outputs a query $(\text{break}, a_1, l_1, \ldots, a_p, l_p, v', M')$, such that (l_1, K_1, v_1) to (l_p, K_p, v_p) are stored, $a_i \in \{0, 1\}$ and no break-query has been sent, compute $K = \sum_{i=0}^{p} a_i K_i$ and $v = \sum_{i=0}^{p} a_i v_i$. If $v' \neq v$ and $M' = \Delta v' + K$ $\mathcal{A}_2$ wins.

Experiment 3 Sample a random global key $\Delta \in \mathbb{F}_q$. $\mathcal{A}_3$ is allowed to send queries (mac, v, l, M), where $v, M \in \mathbb{F}_q$ and l is a fresh label. For each such query, store (l, K, v) and return $K = M - \Delta v$. If $\mathcal{A}_3$ outputs a query (break, Δ'), and no break-query has been sent previously, if $\Delta' = \Delta$, $\mathcal{A}_3$ wins.

Experiment 4 Sample a random global key $\Delta \in \mathbb{F}_q$. If $\mathcal{A}_4$ outputs a query (break, Δ'), and no break-query has been sent previously, if $\Delta' = \Delta$, $\mathcal{A}_4$ wins.

Lemma 5.2. *For every adversary $\mathcal{A}_1$ in Experiment 1 there exists an adversary $\mathcal{A}_2$ that is no stronger than $\mathcal{A}_1$, which wins Experiment 2 with the same success probability as $\mathcal{A}_1$.*

Proof. Given an adversary $\mathcal{A}_1$, $\mathcal{A}_2$ passes all side information on Δ to $\mathcal{A}_1$. For each query (mac, v, l) from $\mathcal{A}_1$, $\mathcal{A}_2$ samples a random MAC $M \in \{0, 1\}_q^{\mathbb{F}}$, outputs (mac, v, l, M) to Experiment 2 and sends M to $\mathcal{A}_1$. If $\mathcal{A}_1$ sends a query $(\text{break}, a_1, l_1, \ldots, a_p, l_p, v', M')$, $\mathcal{A}_2$ forwards it to Experiment 2. The number of queries in both experiments is identical, and the distribution on K and M is identical as well, since in Experiment 1 K is chosen uniformly and $M = \Delta v + K$, while in Experiment 2 M is chosen uniformly and $K = M - \Delta v$. Thus the success probability of $\mathcal{A}_2$ is the same as the success probability of $\mathcal{A}_1$, and $\mathcal{A}_2$'s running time is obviously linear in the running time of $\mathcal{A}_1$. □

Lemma 5.3. *For every adversary $\mathcal{A}_2$ in Experiment 2 there exists an adversary $\mathcal{A}_3$ that is no stronger than $\mathcal{A}_2$, which wins Experiment 3 with the same success probability as $\mathcal{A}_2$.*

Proof. Given an adversary $\mathcal{A}_2$, $\mathcal{A}_3$ passes all side information on Δ to $\mathcal{A}_2$. For each query (mac, v, l, M) from $\mathcal{A}_2$, $\mathcal{A}_3$ outputs this message to Experiment 3 and stores (v, l, M). If $\mathcal{A}_2$ sends a query $(\text{break}, a_1, l_1, \ldots, a_p, l_p, v', M')$, where all tuples (v_i, l_i, M_i), $i \in \{1, \ldots, p\}$ are stored, compute $M = \sum_{i=1}^{p} a_i M_i$ and $v = \sum_{i=1}^{p} a_i v_i$.

For each (l_i, M_i, v_i) let K_i be the corresponding key stored by Experiment 3. It holds that $M_i = \Delta v_i + K_i$, so if we let $K = \sum_{i-1}^{p} a_i K_i$, then $M = \Delta v + K$. If $\mathcal{A}_2$ wins, i.e. $M' = \Delta v' + K$, we can compute $M - M' = \Delta v + K - \Delta v' - K - = \Delta(v - v')$. Dividing by $(v - v')$ yields Δ, so $\mathcal{A}_3$ outputs $(\text{break}, (M - M')/(v - v'))$ and wins Experiment 3. Obviously, $\mathcal{A}_2$ and $\mathcal{A}_3$ have the same success probability. □

Lemma 5.4. *For every adversary $\mathcal{A}_3$ in Experiment 3 there exists an adversary $\mathcal{A}_4$ that is no stronger than $\mathcal{A}_3$, which wins Experiment 4 with the same success probability as $\mathcal{A}_3$.*

Proof. $\mathcal{A}_4$ internally runs $\mathcal{A}_3$ and simply ignores the MAC queries, because they do not influence the success probability of $\mathcal{A}_3$. □

We thus showed that any adversary that can forge a MAC essentially has to guess the global key Δ. Since the simulation is perfect until a MAC is forged and the field size is exponential, it follows that the simulation distributed statistically close to the real protocol. □

6 EFFICIENCY OF OUR APPROACH

We give a short comparison with recent 2PC protocols that follow the same paradigm of implementing an arithmetic black box.

In order to evaluate the efficiency of our approach, we first have to find a suitable instantiation of the OLE primitive. Currently, the very recent result of Ghosh *et al.* [13] based on noisy encodings seems to have the highest practical efficiency both communication-wise and computation-wise. Their noisy encoding is based on a more aggressive assumption of noisy polynomial reconstruction; similar assumptions have been used previously in [17, 21]. From their paper, we can deduce the communication cost in field elements by

$$c_{\text{OLE}} = \alpha(2 + c_{\text{OT}})$$

for a single OLE. Here α is depending on the security parameters (basically the noise rate of the encoding) of the underlying assumption. According to [13] $\alpha = 8$ is a more optimistic choice, and $\alpha = 16$ a more conservative one. The cost of OT c_{OT} can be fixed to 2 field elements (when using OT extension). Assuming $\alpha = 16$, this means we need 64 field elements per OLE. Combining this with the overhead of our protocol we get that an authenticated triple costs

$$c_{\text{triple}} = 2(c_{\text{OLE}} \cdot 11 + 4) = 1416$$

field elements. It might be convenient to implement the OLE and then derandomize some of values for our protocol, because there are some dependencies between the inputs of the OLEs. But this will only induce an small additive overhead to c_{triple} (there are at most 22 values that have to be derandomized per triple), which is insignificant in comparison to the cost of the OLEs. All in all, we can bound the number of field elements by 1440.

Protocol	Security	Field	Comm./Triple
SPDZ [8]	active	$\mathbb{F}_p$, 128-bit	430kBit
	covert	$\mathbb{F}_p$, 128-bit	132kBit
MASCOT [18]	active	$\mathbb{F}_q$, 128-bit	360kBit
	active	$\mathbb{F}_q$, 64-bit	106kBit
This work	active	$\mathbb{F}_q$, 128-bit	180kBit
	active	$\mathbb{F}_q$, 64-bit	90kBit

Table 1: Comparison of communication overhead per authenticated triple with existing solutions (taken from [18]).

Based on the above analysis we estimate the communication overhead of our protocol per authenticated triple and compare this in Table 1 with existing solutions. Looking at those numbers, it is interesting to see that our approach beats previous solutions even for smaller fields or with weaker security guarantees.

We currently have no implementation of the OLE protocol of [13], so we cannot give a fair comparison of the computational overhead with respect to other approaches. But we want to highlight that the OLE protocol only requires basic field operations and polynomial interpolation, which has a computational overhead of $n \log n$ for n OLEs. Our protocol itself only needs a constant number of basic field operations. It therefore seems a reasonable assumption that network bandwidth is the limiting factor for the triple generation, and not the computational overhead. This favors our approach over previous solutions.

REFERENCES

[1] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. 2017. Secure Arithmetic Computation with Constant Computational Overhead. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. 223–254. https://doi.org/10.1007/978-3-319-63688-7_8

[2] Zuzana Beerliová-Trubíniová and Martin Hirt. 2008. Perfectly-Secure MPC with Linear Communication Complexity. In *TCC 2008 (LNCS)*, Ran Canetti (Ed.), Vol. 4948. Springer, Heidelberg, 213–230.

[3] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT 2011 (LNCS)*, Kenneth G. Paterson (Ed.), Vol. 6632. Springer, Heidelberg, 169–188.

[4] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*. IEEE Computer Society Press, 136–145.

[5] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. 2001. Multiparty Computation from Threshold Homomorphic Encryption. In *EUROCRYPT 2001 (LNCS)*, Birgit Pfitzmann (Ed.), Vol. 2045. Springer, Heidelberg, 280–299.

[6] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. 1995. Committed Oblivious Transfer and Private Multi-Party Computation. In *CRYPTO'95 (LNCS)*, Don Coppersmith (Ed.), Vol. 963. Springer, Heidelberg, 110–123.

[7] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. 2010. Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography. In *EUROCRYPT 2010 (LNCS)*, Henri Gilbert (Ed.), Vol. 6110. Springer, Heidelberg, 445–465.

[8] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO 2012 (LNCS)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.), Vol. 7417. Springer, Heidelberg, 643–662.

[9] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. 2012. David & Goliath Oblivious Affine Function Evaluation - Asymptotically Optimal Building Blocks for Universally Composable Two-Party Computation from a Single Untrusted Stateful Tamper-Proof Hardware Token. Cryptology ePrint Archive, Report 2012/135. (2012). http://eprint.iacr.org/2012/135.

[10] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. 2012. Statistically Secure Linear-Rate Dimension Extension for Oblivious Affine Function Evaluation. In *ICITS 12 (LNCS)*, Adam Smith (Ed.), Vol. 7412. Springer, Heidelberg, 111–128. https://doi.org/10.1007/978-3-642-32284-6_7

[11] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. 2013. MiniLEGO: Efficient Secure Two-Party Computation from General Assumptions. In *EUROCRYPT 2013 (LNCS)*, Thomas Johansson and Phong Q. Nguyen (Eds.), Vol. 7881. Springer, Heidelberg, 537–556. https://doi.org/10.1007/978-3-642-38348-9_32

[12] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. 2014. Circuits resilient to additive attacks with applications to secure computation. In *46th ACM STOC*, David B. Shmoys (Ed.). ACM Press, 495–504.

[13] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. 2017. Maliciously Secure Oblivious Linear Function Evaluation with Constant Overhead. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Application of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017*. To appear.

[14] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229.

[15] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO 2003 (LNCS)*, Dan Boneh (Ed.), Vol. 2729. Springer, Heidelberg, 145–161.

[16] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2008. Founding Cryptography on Oblivious Transfer - Efficiently. In *CRYPTO 2008 (LNCS)*, David Wagner (Ed.), Vol. 5157. Springer, Heidelberg, 572–591.

[17] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2009. Secure Arithmetic Computation with No Honest Majority. In *TCC 2009 (LNCS)*, Omer Reingold (Ed.), Vol. 5444. Springer, Heidelberg, 294–314.

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

[19] Yehuda Lindell. 2013. Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries. In *CRYPTO 2013, Part II (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8043. Springer, Heidelberg, 1–17. https://doi.org/10.1007/978-3-642-40084-1_1

[20] Yehuda Lindell and Ben Riva. 2014. Cut-and-Choose Yao-Based Secure Computation in the Online/Offline and Batch Settings. In *CRYPTO 2014, Part II (LNCS)*, Juan A. Garay and Rosario Gennaro (Eds.), Vol. 8617. Springer, Heidelberg, 476–494. https://doi.org/10.1007/978-3-662-44381-1_27

[21] Moni Naor and Benny Pinkas. 1999. Oblivious Transfer and Polynomial Evaluation. In *31st ACM STOC*. ACM Press, 245–254.

[22] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. 2012. A New Approach to Practical Active-Secure Two-Party Computation. In *CRYPTO 2012 (LNCS)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.), Vol. 7417. Springer, Heidelberg, 681–700.

[23] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd FOCS*. IEEE Computer Society Press, 160–164.