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

Nico Döttling[1], Satrajit Ghosh[1], Jesper Buus Nielsen[1], Tobias Nilges[1], and Roberto Trifiletti[1]

Dept. of Computer Science, Aarhus University

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 as an important foundation for efficient actively secure 2PC.

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 [NP99]. 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 standard assumptions both with standalone security and UC-security [IPS09,GNN17]; even with information-theoretic UC-security using tamper proof tokens [DKMQ12].

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 [NNOB12] 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 [Yao82]. 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 [GMW87] 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 has 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 [Yao82,GMW87] 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 [Lin13] 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 [FJN+13] 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 [Lin13] but less efficient for meaningful levels of security because of the large constants hidden by the big-O notation.

In [LR14] 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 [FJN+13]. 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 [GMW87] 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.

In 1995 Crépeau, van de Graaf and Tapp [CvT95] introduced the notion of committed oblivious transfer (COT) which is a variant of OT where the parties are committed to

[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.

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 [IPS08] 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 [DIK10], however, results in fairly large constants being hidden by the big-O notation.

In 2012 Nielsen *et al.* [NNOB12] then published a new approach to OT based 2PC. 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 [GMW87] 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 undetected 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 [NNOB12] how to do efficient actively secure OT extension, improving on the semi-honest secure OT extension from [IKNP03]. 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 [NNOB12] indicate that the approach can be practical for reasonably large circuits.

A common drawback of the garbling and OT based approaches is that it works 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 [IPS08] and [NNOB12] 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.

There is a line of work on implementing actively secure MPC of arithmetic circuits, including [CDN01,DPSZ12,BDOZ11]. These protocols all use some form of homomorphic encryption along with zero-knowledge proofs for proving correct behaviour. The complexity of these protocol 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 solution 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 [NNOB12].

The works [IPS09] and [GIP+14] are more directly comparable to our protocol. In [IPS09] Ishai, Prabhakaran and Sahai instantiates 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 [GIP+14] 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 [IPS09,GIP+14], 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.

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 [IPS08] 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 [NNOB12] 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 [IPS09]. In [IPS09] 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 [IPS08] 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 [IPS09] with the same good practical efficiency of [NNOB12].

Another potential advantage of our protocol over [IPS09] and the other protocols for actively secure arithmetic computation mentioned above is that they use black box access to homomorphic encryption, whereas we use OLE. It seems to be a plausible conjecture that we might at some point be able to construct a protocol for actively secure OLE extension *a la* actively secure OT extension from [NNOB12], i.e., a protocol which given a few seed OLEs can generate any polynomial number of OLEs while using only a few applications of an efficient symmetric primitive per generated OLE. This hope is based on the similarity between OT and OLE. On the other hand there does not seem to be similar support for the hope that we might construct a homomorphic encryption extension, i.e., to implement any polynomial number of accesses to homomorphic encryption given only a few seed applications of homomorphic encryption plus access to for instance a hash function. If one could construct a protocol for efficient actively secure OLE extension, our protocol would immediately yield a very practical protocol, which would outcompete [NNOB12] even if using both protocols to compute a Boolean circuit.

Our Techniques

Our approach is somewhat reminiscent of the approach in [NNOB12]. We start with an arithmetic version of [GMW87]. 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 construct in an offline phase a large number of authenticated triples on random values and then we consume a small constant number of these in the online phase when evaluating an arithmetic gate. This forces the parties to input the right values to all gates and hence gives active security. We essentially generate one authenticated multiplication by running one OLE on random values and then letting the parties commit to their values. We use that OLEs themselves can be seen as commitments to the value x to efficiently compute these commitments. Then similar to [NNOB12] we compute a test on each intermediate value to check that the commitments were computed correctly. We call this technique fingerprinting. These fingerprints are significantly different from the test in [NNOB12] 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 [NNOB12]. Our test in principle has a similar problem with a selective attack, but we can carefully arrange the test such that the sender of a and b needs to guess the random field element x to not get detected. This immediate gives security $|\mathbb{F}|^{-1}$, which is negligible for large enough fields. Developing the new test required developing significantly new techniques which exploit essentially that we are in the arithmetic setting where both parties have a large random input. As a result of

our improved test for correct commitment, we do not need to apply the bucketing technique of [NNOB12] which immediately saves us an asymptotic factor of $O(s/\log(s))$. At the same time our test is simple enough that we get a very practical protocol.

Unfortunately our new test does not translate back to the OT based protocol from [NNOB12], 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 as an important foundation for efficient actively secure arithmetic 2PC.

More Technical Details We will now sketch the technical details of our dealer protocol which produces authenticated multiplication triples using OLEs. In the semi-honest case, the purpose of the dealer-protocol is to provide random triples $(a_{\mathsf{A}}, b_{\mathsf{A}}, c_{\mathsf{A}})$ to A and $(a_{\mathsf{B}}, b_{\mathsf{B}}, c_{\mathsf{B}})$ to B such that the relation

$$(a_{\mathsf{A}} + a_{\mathsf{B}}) \cdot (b_{\mathsf{A}} + b_{\mathsf{A}}) = c_{\mathsf{A}} + c_{\mathsf{B}} \tag{1}$$

holds. We can expand this to

$$a_{\mathsf{A}}b_{\mathsf{A}} + a_{\mathsf{A}}b_{\mathsf{B}} + a_{\mathsf{B}}b_{\mathsf{B}} + a_{\mathsf{A}}b_{\mathsf{A}} = c_{\mathsf{A}} + c_{\mathsf{B}} \tag{2}$$

In the simple semi-honest protocol A chooses the values a_{A} and b_{A} at at random and B the values a_{B} and b_{B}. Notice that the terms $c^l_{\mathsf{A}} = a_{\mathsf{A}}b_{\mathsf{A}}$ and $c^l_{\mathsf{B}} = a_{\mathsf{B}}b_{\mathsf{B}}$ can be computed by A and B respectively. Thus, we need to securely compute the mixed terms $a_{\mathsf{B}}b_{\mathsf{A}}$ and $a_{\mathsf{A}}b_{\mathsf{B}}$. 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 (6) as

$$(a_{\mathsf{A}}b_{\mathsf{A}} - r_{\mathsf{A}} + a_{\mathsf{A}}b_{\mathsf{B}} + r_{\mathsf{B}}) + (a_{\mathsf{B}}b_{\mathsf{B}} - r_{\mathsf{B}} + a_{\mathsf{B}}b_{\mathsf{A}} + r_{\mathsf{A}}) = c_{\mathsf{A}} + c_{\mathsf{B}}. \tag{3}$$

This suggests the following protocol to compute the triples $(a_{\mathsf{A}}, b_{\mathsf{A}}, c_{\mathsf{A}})$ and $(a_{\mathsf{B}}, b_{\mathsf{B}}, c_{\mathsf{B}})$:

1. A and B choose $(a_{\mathsf{A}}, b_{\mathsf{A}}, r_{\mathsf{A}})$ and $(a_{\mathsf{B}}, b_{\mathsf{B}}, r_{\mathsf{B}})$ locally at random.
2. A and B compute $c^i_{\mathsf{A}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(a_{\mathsf{B}}, r_{\mathsf{B}}; b_{\mathsf{A}})$ and $c^i_{\mathsf{B}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(a_{\mathsf{A}}, r_{\mathsf{A}}; b_{\mathsf{B}})$.
3. A sets $c_{\mathsf{A}} = c^l_{\mathsf{A}} - r_{\mathsf{A}} + c^i_{\mathsf{A}}$ and B sets $c_{\mathsf{B}} = c^l_{\mathsf{B}} - r_{\mathsf{B}} + c^i_{\mathsf{B}}$.

Correctness of the protocol follows immediately from Equation (7). Semi-honest privacy of the protocol follows from the fact that c^i_{A} and c^i_{B} 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 $K_{a_{\mathsf{B}}}, K_{b_{\mathsf{B}}}$ and $K_{r_{\mathsf{B}}}$. Likewise, B chooses a global MAC key Δ_{A} and specific keys $K_{a_{\mathsf{A}}}, K_{b_{\mathsf{A}}}$ and $K_{r_{\mathsf{A}}}$. Now, A commits to a_{A} by running $M_{a_{\mathsf{A}}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_{\mathsf{A}}, K_{a_{\mathsf{A}}}; a_{\mathsf{A}})$. This MAC can be opened by A by sending a_{A} and $M_{a_{\mathsf{A}}}$ to B. The remaining MACs $M_{b_{\mathsf{A}}}, M_{r_{\mathsf{A}}}$ and $M_{a_{\mathsf{B}}}, M_{b_{\mathsf{B}}}, M_{r_{\mathsf{B}}}$ are computed likewise.

After the above protocol is finished, A obtains MACs on the values c^l_{A} and c^i_{A} by $M_{c^l_{\mathsf{A}}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_{\mathsf{A}}, K_{c^l_{\mathsf{A}}}; c^l_{\mathsf{A}})$ and $M_{c^i_{\mathsf{B}}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_{\mathsf{A}}, K_{c^i_{\mathsf{A}}}; c^i_{\mathsf{A}})$, where $K_{c^l_{\mathsf{A}}}$ and $K_{c^i_{\mathsf{A}}}$ are sampled on the fly by B. Likewise, B commits to c^l_{B} and c^i_{B} by $M_{c^l_{\mathsf{B}}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_{\mathsf{B}}, K_{c^l_{\mathsf{B}}}; c^l_{\mathsf{B}})$ and $M_{c^i_{\mathsf{B}}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_{\mathsf{B}}, K_{c^i_{\mathsf{B}}}; c^i_{\mathsf{B}})$.

Now, A and B can locally compute MACs to c_A and c_B by using the additively homomorphic properties of the MAC scheme. Specifically, A computes $M_{c_\mathsf{A}} = M_{c^l_\mathsf{A}} - M_{r_\mathsf{A}} + M_{c^i_\mathsf{A}}$ and $K_{c_\mathsf{B}} = K_{c^l_\mathsf{B}} - K_{r_\mathsf{B}} + K_{c^i_\mathsf{B}}$, whereas B computes $M_{c_\mathsf{B}} = M_{c^l_\mathsf{B}} - M_{r_\mathsf{B}} + M_{c^i_\mathsf{B}}$ and $K_{c_\mathsf{A}} = K_{c^l_\mathsf{A}} - K_{r_\mathsf{A}} + K_{c^i_\mathsf{A}}$.

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.

- First, we want to make sure that A inputs the correct value c^l_A to obtain $M_{c^l_\mathsf{A}}$. At the same time, B must not use a different Δ_A than for the other MACs. We compute a check value γ^1 as follows. A and B use another $\mathbb{F}_q$-OLE to compute $\sigma^1_\mathsf{A} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(-K_{a_\mathsf{A}}, K_{c^l_\mathsf{A}} + \gamma^1; b_\mathsf{A})$, where B chooses $\gamma^1 \leftarrow_\$ \mathbb{F}_q$. Now A locally computes $b_\mathsf{A} M_{a_\mathsf{A}} + \sigma^1 - M_{c^l_\mathsf{A}}$, which evaluates to γ^1 if $M_{c^l_\mathsf{A}}$ was computed with the correct c^l_A and Δ_A.
- Secondly, we have to ensure that c^i_A is correctly computed. Another OLE is used to compute $\sigma^2 \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(M_{a_\mathsf{B}}, M_{r_\mathsf{B}}; b_\mathsf{A})$. A locally computes $\sigma^{2\prime} = \Delta_\mathsf{B} c^i_\mathsf{A} + K_{a_\mathsf{B}} b_\mathsf{A} + K_{r_\mathsf{B}}$, and if B used the correct inputs, it holds that $\sigma^{2\prime} = \sigma^2$.
- Now that we know that c^i_A is correct, we have to verify that the MAC $M_{c^i_\mathsf{A}}$ was generated correctly. Towards this, we first observe that we can create $M_{c^i_\mathsf{A}}$ in a different way than described above: $M_{c^i_\mathsf{A}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_\mathsf{A} a_\mathsf{B}, \Delta_\mathsf{A} r_\mathsf{B} + K_{c^i_\mathsf{A}}; b_\mathsf{A})$ yields $M_{c^i_\mathsf{A}} = \Delta_\mathsf{A}(a_\mathsf{B} b_\mathsf{A} + r_\mathsf{B}) + K_{c^i_\mathsf{A}}$. Now we create another check value γ^2 by using an additional authenticated value s_A. A locally computes $d = c^i_\mathsf{A} - s_\mathsf{A}$ and $\gamma^2 = M_{c^i_\mathsf{A}} - M_{s_\mathsf{A}}$, and sends d to B. B locally computes $\Delta_\mathsf{A} d + K_{c^i_\mathsf{A}} - K_{s_\mathsf{A}}$. Again, if A used the correct input for the MAC generation, and B as well, both parties will obtain the same γ^2.
- It remains to bind the input of A to the before authenticated value b_A. We therefore add a final check γ^3: A obtains $\sigma^3_\mathsf{A} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_\mathsf{A}, \sigma^3_\mathsf{B}; b_\mathsf{A})$. Now A locally computes $\gamma^3 = M_{b_\mathsf{A}} - \sigma^3_\mathsf{A}$, while B locally computes $K_{b_\mathsf{A}} - \sigma^3_\mathsf{B}$.

In a final step we add all γ^i to a global check value γ^{glo}. Then A commits to its version of γ^{glo}, B sends it version to A, and A has to unveil. If the values do not match, the protocol is aborted.

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 [Can01]. In the framework, security of a protocol is shown by comparing a real protocol π in the real

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

Authenticated Value $[x]_{\mathsf{A}}$ 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 $M_x = K_x + x\Delta_{\mathsf{A}} \in \mathbb{F}_q$.

Let $[x]_{\mathsf{A}} = (x, K_x, M_x)$. We omit Δ_{A} since it is identical for each value x. To compute $[z]_{\mathsf{A}} = [x]_{\mathsf{A}} + [y]_{\mathsf{A}}$, A computes $[z]_{\mathsf{A}} = (z, K_z, M_z) = (x + y, K_x + K_y, M_x + M_y)$. 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 $K_v = v\Delta_{\mathsf{A}}$. Then $[v]_{\mathsf{A}} = (v, K_v, M_v)$. For a constant c we let $[x]_{\mathsf{A}} + c = [x]_{\mathsf{A}} + [c]_{\mathsf{A}}$ and $[x]_{\mathsf{A}} \cdot c = [xc]_{\mathsf{A}} = (xc, K_x c, M_x c)$.

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

Authenticated Share We let $[x] = [x_{\mathsf{A}}|x_{\mathsf{B}}]$ denote that A and B hold authenticated shares x_{A}, x_{B} such that $x = x_{\mathsf{A}} + x_{\mathsf{B}}$.

To compute $[z] = [z_{\mathsf{A}}|z_{\mathsf{B}}] = [x] + [y]$, A locally computes $z_{\mathsf{A}} = x_{\mathsf{A}} + y_{\mathsf{A}}$ and B locally computes $z_{\mathsf{B}} = x_{\mathsf{B}} + y_{\mathsf{B}}$.

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.

Fig. 1. Notation of authenticated values and shares.

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 in the real protocol. To ensure indistinguishability, there has to exist a simulator that produces the same protocol transcript as 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.

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).

There are several implicit and explicit constructions of OLE based on a variety of assumptions, e.g. [DKMQ12,DPSZ12,KOS16,GNN17]. For our protocol we assume OLE as a (UC-secure) black-box.

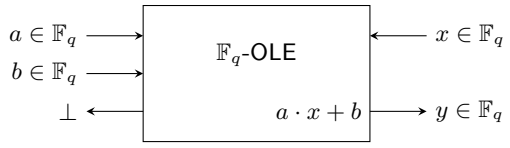

Fig. 2. The $\mathbb{F}_q$-OLE primitive.

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 unconditionally secure commitments. Towards this, Döttling et al. [DKMQ12] present a very simple 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.

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

Lemma 1 ([DKMQ12]). *Protocol Π_{Com} UC-realizes $\mathcal{F}_{\mathsf{Com}}$ in the $\mathbb{F}_q$-OLE-hybrid model.*

The binding property directly follows from the receiver privacy of the OLE, while the hiding property follows from the sender privacy. See [DKMQ12] 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.

Döttling *et al.* [DKM12] 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

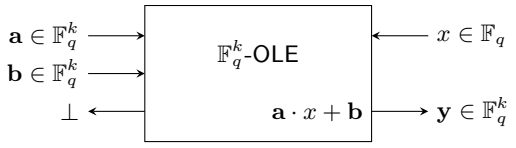

Fig. 4. The $\mathbb{F}_q^k$-OLE primitive.

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

$$\sum_{i=1}^{k} t_i - t_{k+1} = \gamma = \sum_{i=1}^{k} r_i - r_{k+1}, \tag{4}$$

where the sender locally computes the LHS of Equation (4) and the receiver the RHS of Equation (4). 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 $\mathbf{y}$.

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

Lemma 2. *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.*

Proof (Sketch.). **Corrupted Sender.** Assume that $\mathcal{A}_\mathsf{S}$ uses $a_{k+1} \neq \sum_{i=1}^{k} a_i$. 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 $\mathcal{A}_{\mathsf{S}}$ 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. Assume w.l.o.g. that $\mathcal{A}_{\mathsf{R}}$ 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 $\mathcal{A}_{\mathsf{R}}$ 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 $1 - 1/|\mathbb{F}_q|$.

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 $\mathcal{F}_{\mathsf{Deal}}$ realized by this protocol is given in Figure 6.

We will start with a high level overview of our dealer protocol (cf. Figure 8). In the semi-honest case, the purpose of the dealer-protocol is to provide random triples $(a_{\mathsf{A}}, b_{\mathsf{A}}, c_{\mathsf{A}})$ to A and $(a_{\mathsf{B}}, b_{\mathsf{B}}, c_{\mathsf{B}})$ to B such that the relation

$$(a_{\mathsf{A}} + a_{\mathsf{B}}) \cdot (b_{\mathsf{A}} + b_{\mathsf{B}}) = c_{\mathsf{A}} + c_{\mathsf{B}} \tag{5}$$

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

Initialize: On input (`init`) from A and B, sample $\Delta_{\mathsf{A}}, \Delta_{\mathsf{B}} \leftarrow_{\$} \mathbb{F}_q$, $\mathbf{v}_{\mathsf{A}}, \mathbf{v}_{\mathsf{B}}, \mathbf{K}_{\mathbf{v}_{\mathsf{A}}}, \mathbf{K}_{\mathbf{v}_{\mathsf{B}}} \leftarrow_{\$} \mathbb{F}_q^n$ and set $\mathbf{M}_{\mathbf{v}_{\mathsf{A}}} = \Delta_{\mathsf{A}}\mathbf{v}_{\mathsf{A}} + \mathbf{K}_{\mathbf{v}_{\mathsf{A}}}$ and $\mathbf{M}_{\mathbf{v}_{\mathsf{B}}} = \Delta_{\mathsf{B}}\mathbf{v}_{\mathsf{B}} + \mathbf{K}_{\mathbf{v}_{\mathsf{B}}}$. Store $(\Delta_{\mathsf{A}}, \Delta_{\mathsf{B}}, \mathbf{v}_{\mathsf{A}}, \mathbf{v}_{\mathsf{B}}, \mathbf{K}_{\mathbf{v}_{\mathsf{A}}}, \mathbf{K}_{\mathbf{v}_{\mathsf{B}}}, \mathbf{M}_{\mathbf{v}_{\mathsf{A}}}, \mathbf{M}_{\mathbf{v}_{\mathsf{B}}})$ and output Δ_{B} to A and Δ_{A} to B.
Adversarial access: On input $(\texttt{setInit}, \mathsf{A}, (\Delta_{\mathsf{B}}, \mathbf{v}_{\mathsf{A}}, \mathbf{M}_{\mathbf{v}_{\mathsf{A}}}, \mathbf{K}_{\mathbf{v}_{\mathsf{B}}}))$, sample $\Delta_{\mathsf{A}} \leftarrow_{\$} \mathbb{F}_q, \mathbf{v}_{\mathsf{B}} \leftarrow_{\$} \mathbb{F}_q^n$ and set $\mathbf{M}_{\mathbf{v}_{\mathsf{B}}} = \Delta_{\mathsf{B}}\mathbf{v}_{\mathsf{B}} + \mathbf{K}_{\mathbf{v}_{\mathsf{B}}}$ and $\mathbf{K}_{\mathbf{v}_{\mathsf{A}}} = \mathbf{M}_{\mathbf{v}_{\mathsf{A}}} - \Delta_{\mathsf{A}}\mathbf{v}_{\mathsf{A}}$. Symmetrically for a malicious B.

Authenticated Value(A): On input $(\texttt{aValue}, \mathsf{A})$ 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 $a_{\mathsf{A}} = \mathbf{v}_{\mathsf{A}}[i_1], b_{\mathsf{A}} = \mathbf{v}_{\mathsf{A}}[i_2], a_{\mathsf{B}} = \mathbf{v}_{\mathsf{B}}[i_1], b_{\mathsf{B}} = \mathbf{v}_{\mathsf{B}}[i_2]$. Pick $c_{\mathsf{A}}, c_{\mathsf{B}} \leftarrow_{\$} \mathbb{F}_q$ under the restriction that $c_{\mathsf{A}} + c_{\mathsf{B}} = (a_{\mathsf{A}} + a_{\mathsf{B}})(b_{\mathsf{A}} + b_{\mathsf{B}})$. Sample $K_{c_{\mathsf{A}}}, K_{c_{\mathsf{B}}} \leftarrow_{\$} \mathbb{F}_q$ and set $M_{c_{\mathsf{A}}} = \Delta_{\mathsf{A}}c_{\mathsf{A}} + K_{c_{\mathsf{A}}}$ and $M_{c_{\mathsf{B}}} = \Delta_{\mathsf{B}}c_{\mathsf{B}} + K_{c_{\mathsf{B}}}$. Output $(a_{\mathsf{A}}, b_{\mathsf{A}}, c_{\mathsf{A}}, M_{a_{\mathsf{A}}}, M_{b_{\mathsf{A}}}, M_{c_{\mathsf{A}}}, K_{a_{\mathsf{B}}}, K_{b_{\mathsf{B}}}, K_{c_{\mathsf{B}}})$ to A, symmetrically to B.
Adversarial access: On input $(\texttt{setMult}, \mathsf{A}, (c_{\mathsf{A}}, M_{c_{\mathsf{A}}}, K_{c_{\mathsf{B}}}))$ pick $c_{\mathsf{B}} \leftarrow_{\$} \mathbb{F}_q$ under the restriction that $c_{\mathsf{A}} + c_{\mathsf{B}} = (a_{\mathsf{A}} + a_{\mathsf{B}})(b_{\mathsf{A}} + b_{\mathsf{B}})$. Set $M_{c_{\mathsf{B}}} = \Delta_{\mathsf{B}}c_{\mathsf{B}} + K_{c_{\mathsf{B}}}$ and $K_{c_{\mathsf{A}}} = M_{c_{\mathsf{A}}} - \Delta_{\mathsf{A}}c_{\mathsf{A}}$. Symmetrically for a malicious B.

Fig. 6. Ideal dealer functionality.

holds. We can expand this to

$$a_{\mathsf{A}}b_{\mathsf{A}} + a_{\mathsf{A}}b_{\mathsf{B}} + a_{\mathsf{B}}b_{\mathsf{B}} + a_{\mathsf{B}}b_{\mathsf{A}} = c_{\mathsf{A}} + c_{\mathsf{B}} \ . \tag{6}$$

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 $c^l_{\mathsf{A}} = a_{\mathsf{A}}b_{\mathsf{A}}$ and $c^l_{\mathsf{B}} = a_{\mathsf{B}}b_{\mathsf{B}}$ can be locally computed by A and B respectively. Thus, we need to securely compute the mixed terms $a_{\mathsf{B}}b_{\mathsf{A}}$ and $a_{\mathsf{A}}b_{\mathsf{B}}$. 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 (6) as

$$(a_{\mathsf{A}}b_{\mathsf{A}} - r_{\mathsf{A}} + a_{\mathsf{A}}b_{\mathsf{B}} + r_{\mathsf{B}}) + (a_{\mathsf{B}}b_{\mathsf{B}} - r_{\mathsf{B}} + a_{\mathsf{B}}b_{\mathsf{A}} + r_{\mathsf{A}}) = c_{\mathsf{A}} + c_{\mathsf{B}} \ . \tag{7}$$

This suggests the following protocol to compute the triples $(a_{\mathsf{A}}, b_{\mathsf{A}}, c_{\mathsf{A}})$ and $(a_{\mathsf{B}}, b_{\mathsf{B}}, c_{\mathsf{B}})$:

1. A and B choose $(a_{\mathsf{A}}, b_{\mathsf{A}}, r_{\mathsf{A}})$ and $(a_{\mathsf{B}}, b_{\mathsf{B}}, r_{\mathsf{B}})$ locally at random.
2. A and B compute $c^i_{\mathsf{A}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(a_{\mathsf{B}}, r_{\mathsf{B}}; b_{\mathsf{A}})$ and $c^i_{\mathsf{B}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(a_{\mathsf{A}}, r_{\mathsf{A}}; b_{\mathsf{B}})$.
3. A sets $c_{\mathsf{A}} = c^l_{\mathsf{A}} - r_{\mathsf{A}} + c^i_{\mathsf{A}}$ and B sets $c_{\mathsf{B}} = c^l_{\mathsf{B}} - r_{\mathsf{B}} + c^i_{\mathsf{B}}$.

Correctness of the protocol follows immediately from Equation (7). Semi-honest privacy of the protocol follows from the fact that c^i_{A} and c^i_{B} 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 $K_{a_{\mathsf{B}}}, K_{b_{\mathsf{B}}}$ and $K_{r_{\mathsf{B}}}$. Likewise, B chooses a global MAC key Δ_{A} and specific keys $K_{a_{\mathsf{A}}}, K_{b_{\mathsf{A}}}$ and $K_{r_{\mathsf{A}}}$. Now, A commits to a_{A} by running $M_{a_{\mathsf{A}}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_{\mathsf{A}}, K_{a_{\mathsf{A}}}; a_{\mathsf{A}})$. This MAC can be opened by A by sending a_{A} and $M_{a_{\mathsf{A}}}$ to B. The remaining MACs $(M_{b_{\mathsf{A}}}, M_{r_{\mathsf{A}}})$ and $(M_{a_{\mathsf{B}}}, M_{b_{\mathsf{B}}}, M_{r_{\mathsf{B}}})$ are computed likewise (cf. also Figure 7).

After the above protocol is finished, A obtains MACs on the values c^l_{A} and c^i_{A} by $M_{c^l_{\mathsf{A}}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_{\mathsf{A}}, K_{c^l_{\mathsf{A}}}; c^l_{\mathsf{A}})$ and $M_{c^i_{\mathsf{A}}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_{\mathsf{A}}, K_{c^i_{\mathsf{A}}}; c^i_{\mathsf{A}})$, where $K_{c^l_{\mathsf{A}}}$ and $K_{c^i_{\mathsf{A}}}$ are sampled

on the fly by B. Likewise, B commits to c^l_B and c^i_B by $M_{c^l_\mathsf{B}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_\mathsf{B}, K_{c^l_\mathsf{B}}; c^l_\mathsf{B})$ and $M_{c^i_\mathsf{B}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_\mathsf{B}, K_{c^i_\mathsf{B}}; c^i_\mathsf{B})$.

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 $M_{c_\mathsf{A}} = M_{c^l_\mathsf{A}} - M_{r_\mathsf{A}} + M_{c^i_\mathsf{A}}$ and $K_{c_\mathsf{B}} = K_{c^l_\mathsf{B}} - K_{r_\mathsf{B}} + K_{c^i_\mathsf{B}}$, whereas B computes $M_{c_\mathsf{B}} = M_{c^l_\mathsf{B}} - M_{r_\mathsf{B}} + M_{c^i_\mathsf{B}}$ and $K_{c_\mathsf{A}} = K_{c^l_\mathsf{A}} - K_{r_\mathsf{A}} + K_{c^i_\mathsf{A}}$.

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.

- First, we want to make sure that A inputs the correct value c^l_A to obtain $M_{c^l_\mathsf{A}}$. At the same time, B must not use a different Δ_A than for the other MACs. We compute a check value γ^1 as follows. A and B use another $\mathbb{F}_q$-OLE to compute $\sigma^1_\mathsf{A} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(-K_{a_\mathsf{A}}, K_{c^l_\mathsf{A}} + \gamma^1; b_\mathsf{A})$, where B chooses $\gamma^1 \leftarrow_\$ \mathbb{F}_q$. Now A locally computes $b_\mathsf{A} M_{a_\mathsf{A}} + \sigma^1 - M_{c^l_\mathsf{A}}$, which evaluates to γ^1 if $M_{c^l_\mathsf{A}}$ was computed with the correct c^l_A and Δ_A.
- Secondly, we have to ensure that c^i_A is correctly computed. Another OLE is used to compute $\sigma^2 \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(M_{a_\mathsf{B}}, M_{r_\mathsf{B}}; b_\mathsf{A})$. A locally computes $\sigma^{2\prime} = \Delta_\mathsf{B} c^i_\mathsf{A} + K_{a_\mathsf{B}} b_\mathsf{A} + K_{r_\mathsf{B}}$, and if B used the correct inputs, it holds that $\sigma^{2\prime} = \sigma^2$.
- Now that we know that c^i_A is correct, we have to verify that the MAC $M_{c^i_\mathsf{A}}$ was generated correctly. Towards this, we first observe that we can create $M_{c^i_\mathsf{A}}$ in a different way than described above: $M_{c^i_\mathsf{A}} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_\mathsf{A} a_\mathsf{B}, \Delta_\mathsf{A} r_\mathsf{B} + K_{c^i_\mathsf{A}}; b_\mathsf{A})$ yields $M_{c^i_\mathsf{A}} = \Delta_\mathsf{A}(a_\mathsf{B} b_\mathsf{A} + r_\mathsf{B}) + K_{c^i_\mathsf{A}}$. Now we create another check value γ^2 by using an additional authenticated value s_A. A locally computes $d = c^i_\mathsf{A} - s_\mathsf{A}$ and $\gamma^2 = M_{c^i_\mathsf{A}} - M_{s_\mathsf{A}}$, and sends d to B. B locally computes $\Delta_\mathsf{A} d + K_{c^i_\mathsf{A}} - K_{s_\mathsf{A}}$. Again, if A used the correct input for the MAC generation, and B as well, both parties will obtain the same γ^2.
- It remains to bind the input of A to the before authenticated value b_A. We therefore add a final check γ^3: A obtains $\sigma^3_\mathsf{A} \leftarrow \mathbb{F}_q\text{-}\mathsf{OLE}(\Delta_\mathsf{A}, \sigma^3_\mathsf{B}; b_\mathsf{A})$. Now A locally computes $\gamma^3 = M_{b_\mathsf{A}} - \sigma^3_\mathsf{A}$, while B locally computes $K_{b_\mathsf{A}} - \sigma^3_\mathsf{B}$.

In a final step we add all γ^i to a global check value γ^{glo}. Then A commits to its version of γ^{glo}, B sends its version 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 $\mathbb{F}^k_q$-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.

Theorem 1. *The protocol Π_{Deal} UC-realizes $\mathcal{F}_{\mathsf{Deal}}$ in the ($\mathbb{F}_q$-OLE, $\mathbb{F}^k_q$-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 $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ plays both sender and receiver in a complete execution of Π_{Deal}). This wrapper simulator interacts with $\mathcal{F}_{\mathsf{Deal}}$ and provides an indistinguishable simulation.

Π_{Deal}

Initialize: On input (`init`), A (resp. B) samples $\Delta_{\mathsf{B}} \in \mathbb{F}_q$ (resp. Δ_{A}) uniformly at random and stores 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 $\Delta_{\mathsf{S}}\mathbf{v} + \mathbf{r}$. Send $\mathbf{w} = \mathbf{v}\Delta_{\mathsf{S}} + \mathbf{r} + \mathbf{K}$ 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)

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

The wrapper simulator $\mathcal{S}_{\mathsf{wrap}}$ (cf. Figure 9) first runs the initialization phase, where a complete state is created, i.e. all inputs of $\mathcal{A}_{\mathsf{P}}$ 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 $\mathcal{A}_{\mathsf{P}}$'s secrets.

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

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

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

Corrupted sender. The simulator $\mathcal{S}_{\mathsf{S}}$ (cf. Figure 10) creates a complete set of keys for the initialization phase and extracts the corresponding MACs and inputs of $\mathcal{A}_{\mathsf{S}}$. 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 $\mathcal{A}_{\mathsf{S}}$'s input with the correct input and abort if they do not match.

Let Δ_{R}, $(\mathbf{K}_{\mathbf{a}_{\mathsf{R}}}, \mathbf{K}_{\mathbf{b}_{\mathsf{R}}}, \mathbf{K}_{\mathbf{r}_{\mathsf{R}}}, \mathbf{K}_{\mathbf{s}_{\mathsf{R}}})$ be $\mathcal{A}_{\mathsf{S}}$'s input to the first $\mathbb{F}_q^{4n}$-OLE and $(\mathbf{a}_{\mathsf{S}}, \mathbf{b}_{\mathsf{S}}, \mathbf{r}_{\mathsf{S}}, \mathbf{s}_{\mathsf{S}})$ the input and $(\mathbf{M}_{\mathbf{a}_{\mathsf{S}}}, \mathbf{M}_{\mathbf{b}_{\mathsf{S}}}, \mathbf{M}_{\mathbf{r}_{\mathsf{S}}}, \mathbf{M}_{\mathbf{s}_{\mathsf{S}}})$ 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 $M_{c^i_{\mathsf{S}}}$ (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 $\mathcal{A}_{\mathsf{S}}$'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 $\mathcal{A}_{\mathsf{S}}$ to R. If $(\xi_1, \xi_2, \xi_3) \neq (a_{\mathsf{S}} b_{\mathsf{S}}, b_{\mathsf{S}}, d_{\mathsf{S}})$, set $\gamma^{\mathsf{glo}}_{\mathsf{R}} \leftarrow_{\$} \mathbb{F}_q$.

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

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

$$r_{\mathsf{R}} = c^i_{\mathsf{S}} - a_{\mathsf{R}}\xi_2 \tag{8}$$

$$K_{a_{\mathsf{S}}} = M_{a_{\mathsf{S}}} - \Delta_{\mathsf{S}} a_{\mathsf{S}} \tag{9}$$

$$K_{b_{\mathsf{S}}} = M_{b_{\mathsf{S}}} - \Delta_{\mathsf{S}} b_{\mathsf{S}} \tag{10}$$

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 $[a]_\mathsf{S}, [b]_\mathsf{S}, [r]_\mathsf{S}, [s]_\mathsf{S}$.

Multiplication of local shares: The result is $[c^l]_\mathsf{S} = [a_\mathsf{S} b_\mathsf{S}]_\mathsf{S}$.

1. S: Compute $c^l_\mathsf{S} = a_\mathsf{S} \cdot b_\mathsf{S}$. Input c^l_S into $\mathbb{F}_q$-OLE.
2. R: Draw $K_{c^l_\mathsf{S}} \in \mathbb{F}_q$ uniformly at random, input $(\Delta_\mathsf{S}, K_{c^l_\mathsf{S}})$ into $\mathbb{F}_q$-OLE and output $K_{c^l_\mathsf{S}}$.
3. S: Obtain $M_{c^l_\mathsf{S}} = \Delta_\mathsf{S} c^l_\mathsf{S} + K_{c^l_\mathsf{S}}$ and output $(c^l_\mathsf{S}, M_{c^l_\mathsf{S}})$.

Multiplication of cross shares: The result is $[c^i]_\mathsf{S} = [a_\mathsf{R} b_\mathsf{S} + r_\mathsf{R}]_\mathsf{S}$.

1. R: Draw $K_{c^i_\mathsf{S}} \in \mathbb{F}_q$ uniformly at random and input $\big((a_\mathsf{R}, \Delta_\mathsf{S} a_\mathsf{R}, -K_{a_\mathsf{S}}, M_{a_\mathsf{R}}, \Delta_\mathsf{S}), (r_\mathsf{R}, \Delta_\mathsf{S} r_\mathsf{R} + K_{c^i_\mathsf{S}}, K_{c^l_\mathsf{S}} + \gamma^1_\mathsf{R}, M_{r_\mathsf{R}}, \sigma^3_\mathsf{R})\big)$ into $\mathbb{F}^5_q$-OLE.
2. S: Input b_S into $\mathbb{F}^5_q$-OLE and obtain the following values:
   - $c^i_\mathsf{S} = a_\mathsf{R} b_\mathsf{S} + r_\mathsf{R}$
   - $M_{c^i_\mathsf{S}} = \Delta_\mathsf{S}(a_\mathsf{R} b_\mathsf{S} + r_\mathsf{R}) + K_{c^i_\mathsf{S}}$
   - $\sigma^1_\mathsf{S} = -b_\mathsf{S} K_{a_\mathsf{S}} + K_{c^l_\mathsf{S}} + \gamma^1_\mathsf{R}$
   - $\sigma^2_\mathsf{S} = M_{a_\mathsf{R}} b_\mathsf{S} + M_{r_\mathsf{R}} = \Delta_\mathsf{R}(r_\mathsf{R} + b_\mathsf{S} a_\mathsf{R}) + K_{r_\mathsf{R}} + K_{a_\mathsf{R}} b_\mathsf{S}$
   - $\sigma^3_\mathsf{S} = \Delta_\mathsf{S} b_\mathsf{S} + \sigma^3_\mathsf{R}$

   Output $(c^i_\mathsf{S}, M_{c^i_\mathsf{S}})$.

Checks and output: The result is $[a_\mathsf{S}]_\mathsf{S}, [b_\mathsf{S}]_\mathsf{S}, [c_\mathsf{S}]_\mathsf{S}$. Let $\gamma^{\mathsf{glo}}_\mathsf{P}$ for $\mathsf{P} \in \{\mathsf{S}, \mathsf{R}\}$ be the global checksum.

1. Verification of $M_{c^l_\mathsf{S}}$.
   - S: Compute $\gamma^1_\mathsf{S} = b_\mathsf{S} M_{a_\mathsf{S}} + \sigma^1_\mathsf{S} - M_{c^l_\mathsf{S}}$ and set $\gamma^{\mathsf{final}}_\mathsf{S} \mathrel{+}= \gamma^1_\mathsf{S}$.
   - R: Set $\gamma^{\mathsf{glo}}_\mathsf{R} \mathrel{+}= \gamma^1_\mathsf{R}$.
2. Verification of c^i_S.
   - S: Compute $\sigma^{2'}_\mathsf{S} = \Delta_\mathsf{R} c^i_\mathsf{S} + K_{a_\mathsf{R}} b_\mathsf{S} + K_{r_\mathsf{R}}$. Abort if $\sigma^{2'}_\mathsf{S} \neq \sigma^2_\mathsf{S}$. Further set $\gamma^3_\mathsf{S} = M_{b_\mathsf{S}} - \sigma^3_\mathsf{S}$ and $\gamma^{\mathsf{glo}}_\mathsf{S} \mathrel{+}= \gamma^3_\mathsf{S}$.
   - R: Compute $\gamma^3_\mathsf{R} = K_{b_\mathsf{S}} - \sigma^3_\mathsf{R}$ and set $\gamma^{\mathsf{glo}}_\mathsf{S} \mathrel{+}= \gamma^3_\mathsf{R}$.
3. Verification of $M_{c^i_\mathsf{S}}$.
   - S: Compute $\gamma^2_\mathsf{S} = M_{c^i_\mathsf{S}} - M_{s_\mathsf{S}}$ and set $\gamma^{\mathsf{glo}}_\mathsf{S} \mathrel{+}= \gamma^2_\mathsf{S}$. Send $d_\mathsf{S} = c^i_\mathsf{S} - s_\mathsf{S}$ to R.
   - R: Compute $\gamma^2_\mathsf{R} = \Delta_\mathsf{S} d_\mathsf{S} + K_{c^i_\mathsf{S}} - K_{s_\mathsf{S}}$ and set $\gamma^{\mathsf{glo}}_\mathsf{R} \mathrel{+}= \gamma^2_\mathsf{R}$.
4. Final check over all n triples:
   - S: Compute $(\mathsf{com}, \mathsf{unv}) = \mathsf{COM.Commit}(\gamma^{\mathsf{glo}}_\mathsf{S})$ and send com to R.
   - R: Send $\gamma^{\mathsf{glo}}_\mathsf{R}$ to S.
   - S: If $\gamma^{\mathsf{glo}}_\mathsf{R} \neq \gamma^{\mathsf{glo}}_\mathsf{S}$, abort. Otherwise send unv to R, set $[c_\mathsf{S}]^{(j)}_\mathsf{S} = [c^l]^{(j)}_\mathsf{S} - [r]^{(j)}_\mathsf{S} + [c^i]^{(j)}_\mathsf{S}$ and output $(a^{(j)}_\mathsf{S}, M^{(j)}_{a_\mathsf{S}}, b^{(j)}_\mathsf{S}, M^{(j)}_{b_\mathsf{S}}, c^{(j)}_\mathsf{S}, M^{(j)}_{c_\mathsf{S}})$ for $j \in [n]$.
   - R: If $\gamma^{\mathsf{glo}}_\mathsf{R} \neq \gamma^{\mathsf{glo}}_\mathsf{S}$, abort, otherwise output $(K^{(j)}_{a_\mathsf{S}}, K^{(j)}_{b_\mathsf{S}}, K^{(j)}_{c_\mathsf{S}})$ for $j \in [n]$.

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

$$K_{s_\mathsf{S}} = M_{s_\mathsf{S}} - \Delta_\mathsf{S} s_\mathsf{S} \tag{11}$$

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

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

Authenticated value: Simulate according to Π_{Deal}.

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

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

Simulator $\mathcal{S}_{\mathsf{S}}$

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

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

Authenticated value: Simulate according to Π_{Deal}.

Authenticated multiplication ($\mathcal{S}_{\mathsf{mul}}$): Simulate Π_{Deal} and let ξ_1 be $\mathcal{A}_{\mathsf{S}}$'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 $\mathcal{A}_{\mathsf{S}}$.

- If $\xi_1 \neq a_{\mathsf{S}} b_{\mathsf{S}}$ and $\xi_2 \neq b_{\mathsf{S}}$, set $\gamma^1_{\mathsf{R}} \leftarrow_{\$} \mathbb{F}_q$.
- If $\xi_2 \neq b_{\mathsf{S}}$, set $\gamma^3_{\mathsf{R}} \leftarrow_{\$} \mathbb{F}_q$.
- If $\xi_3 \neq c^i_{\mathsf{S}} - s_{\mathsf{S}}$, set $\gamma^2_{\mathsf{R}} \leftarrow_{\$} \mathbb{F}_q$.

Continue the simulation according to Π_{Deal} and compute $c_{\mathsf{S}} = c^l_{\mathsf{S}} + c^i_{\mathsf{S}} - r_{\mathsf{S}}$ and $M_{c_{\mathsf{S}}} = M_{c^l_{\mathsf{S}}} + M_{c^i_{\mathsf{S}}} - M_{r_{\mathsf{S}}}$ if no abort occurs. Output $(c_{\mathsf{S}}, M_{c_{\mathsf{S}}})$.

Fig. 10. Simulator $\mathcal{S}_{\mathsf{S}}$ against a corrupted sender.

$$K_{c^l_{\mathsf{S}}} = M_{c^l_{\mathsf{S}}} - \Delta_{\mathsf{S}}\xi_1 \tag{12}$$

$$K_{c^i_{\mathsf{S}}} = M_{c^i_{\mathsf{S}}} - \Delta_{\mathsf{S}}(a_{\mathsf{R}}\xi_2 + r_{\mathsf{R}}) = M_{c^i_{\mathsf{S}}} - \Delta_{\mathsf{S}}c^i_{\mathsf{S}} \tag{13}$$

$$\begin{aligned}\gamma^1_{\mathsf{R}} &= \sigma^1_{\mathsf{S}} + K_{a_{\mathsf{S}}}\xi_2 - K_{c^l_{\mathsf{S}}} \\ &= \sigma^1_{\mathsf{S}} + (M_{a_{\mathsf{S}}} - \Delta_{\mathsf{S}}a_{\mathsf{S}})\xi_2 - (M_{c^l_{\mathsf{S}}} - \Delta_{\mathsf{S}}\xi_1) \\ &= \sigma^1_{\mathsf{S}} + \Delta_{\mathsf{S}}(\xi_1 - a_{\mathsf{S}}\xi_2) + \xi_2 M_{a_{\mathsf{S}}} - M_{c^l_{\mathsf{S}}}\end{aligned} \tag{14}$$

$$\sigma^3_{\mathsf{R}} = \sigma^3_{\mathsf{S}} - \Delta_{\mathsf{S}}\xi_2 \tag{15}$$

Thus, all of these values are uniformly distributed given $\mathcal{A}_{\mathsf{S}}$'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 $(\xi_1, \xi_2, \xi_3) = (a_{\mathsf{S}}b_{\mathsf{S}}, b_{\mathsf{S}}, d_{\mathsf{S}})$, 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 $(\xi_1, \xi_2, \xi_3) \neq (a_{\mathsf{S}}b_{\mathsf{S}}, b_{\mathsf{S}}, d_{\mathsf{S}})$ and pass the checks of γ^{glo}.

The abort condition for the check γ^1 can be rewritten as

$$\begin{aligned}0 &= b_{\mathsf{S}}M_{a_{\mathsf{S}}} + \sigma^1_{\mathsf{S}} - M_{c^l_{\mathsf{S}}} - \gamma^1_{\mathsf{R}} \\ &= b_{\mathsf{S}}M_{a_{\mathsf{S}}} + \sigma^1_{\mathsf{S}} - M_{c^l_{\mathsf{S}}} - (\sigma^1_{\mathsf{S}} + \Delta_{\mathsf{S}}(\xi_1 - a_{\mathsf{S}}\xi_2) + \xi_2 M_{a_{\mathsf{S}}} - M_{c^l_{\mathsf{S}}}) \\ &= \Delta_{\mathsf{S}}(a_{\mathsf{S}}\xi_2 - \xi_1) + M_{a_{\mathsf{S}}}(b_{\mathsf{S}} - \xi_2)\end{aligned} \tag{16}$$

using Equation (14). By applying Equations (11) and (13) we can rewrite the check of γ^2 as

$$\begin{aligned} 0 &= M_{c_{\mathsf{S}}^i} - M_{s_{\mathsf{S}}} - \Delta_{\mathsf{S}}\xi_3 - K_{c_{\mathsf{S}}^i} + K_{s_{\mathsf{S}}} \\ &= M_{c_{\mathsf{S}}^i} - M_{s_{\mathsf{S}}} - \Delta_{\mathsf{S}}\xi_3 - M_{c_{\mathsf{S}}^i} + \Delta_{\mathsf{S}} c_{\mathsf{S}}^i + M_{s_{\mathsf{S}}} - \Delta_{\mathsf{S}} s_{\mathsf{S}} \\ &= \Delta_{\mathsf{S}}(c_{\mathsf{S}}^i - s_{\mathsf{S}} - \xi_3) \end{aligned} \tag{17}$$

Finally, for the check γ^3 we get

$$\begin{aligned} 0 &= M_{b_{\mathsf{S}}} - \sigma_{\mathsf{S}}^3 - K_{b_{\mathsf{S}}} + \sigma_{\mathsf{R}}^3 \\ &= M_{b_{\mathsf{S}}} - \sigma_{\mathsf{S}}^3 - M_{b_{\mathsf{S}}} + \Delta_{\mathsf{S}} b_{\mathsf{S}} + \sigma_{\mathsf{S}}^3 - \Delta_{\mathsf{S}}\xi_2 \\ &= \Delta_{\mathsf{S}}(b_{\mathsf{S}} - \xi_2). \end{aligned} \tag{18}$$

by applying Equations (10) and (15).

Clearly, if $\xi_1 = a_{\mathsf{S}} b_{\mathsf{S}}$ and $\xi_2 = b_{\mathsf{S}}$, Equations (16) and (18) hold. But if $\xi_1 \neq a_{\mathsf{S}} b_{\mathsf{S}}$, then Equation (16) is uniformly distributed given $\mathcal{A}_{\mathsf{S}}$'s view and the check of γ^1 will fail. On the other hand, if $\xi_2 \neq b_{\mathsf{S}}$, then Equation (18) is uniformly distributed, and the check for γ^3 fails.

Similarly, if $\xi_3 \neq c_{\mathsf{S}}^i - s_{\mathsf{S}}$, γ_{R}^2 will be uniformly distributed given $\mathcal{A}_{\mathsf{S}}$'s view. Thus, if $(\xi_1, \xi_2, \xi_3) \neq (a_{\mathsf{S}} b_{\mathsf{S}}, b_{\mathsf{S}}, c_{\mathsf{S}}^i - s_{\mathsf{S}})$, the probability that $\mathcal{A}_{\mathsf{S}}$ passes the check for γ^{glo} is upper bounded by $1/|\mathbb{F}_q|$, since $\gamma^{\mathsf{glo}} = \gamma^1 + \gamma^2 + \gamma^3$. 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 $n/|\mathbb{F}_q|$, which is negligible in the security parameter. This establishes a correct simulation against a corrupted sender in Π_{Deal}.

Corrupted receiver. The simulator $\mathcal{S}_{\mathsf{R}}$ against a corrupted receiver $\mathcal{A}_{\mathsf{R}}$ is conceptually very simple. During the initialization, it simply extracts the global MAC key and all other keys from $\mathcal{A}_{\mathsf{R}}$ 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 $\mathcal{A}_{\mathsf{R}}$. 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.

Let Δ_{S}, $(\mathbf{K}_{\mathbf{a}_{\mathsf{S}}}, \mathbf{K}_{\mathbf{b}_{\mathsf{S}}}, \mathbf{K}_{\mathbf{r}_{\mathsf{S}}}, \mathbf{K}_{\mathbf{s}_{\mathsf{S}}})$ be $\mathcal{A}_{\mathsf{R}}$'s input to the first $\mathbb{F}_q^{4n}$-OLE and $(\mathbf{a}_{\mathsf{R}}, \mathbf{b}_{\mathsf{R}}, \mathbf{r}_{\mathsf{R}}, \mathbf{s}_{\mathsf{R}})$ the input and $(\mathbf{M}_{\mathbf{a}_{\mathsf{R}}}, \mathbf{M}_{\mathbf{b}_{\mathsf{R}}}, \mathbf{M}_{\mathbf{r}_{\mathsf{R}}}, \mathbf{M}_{\mathbf{s}_{\mathsf{R}}})$ the output of the second $\mathbb{F}_q^{4n}$-OLE. Further let (α_1, β_1) be $\mathcal{A}_{\mathsf{R}}$'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 $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_{\mathsf{R}}, M_{a_{\mathsf{R}}}, r_{\mathsf{R}}, M_{a_{\mathsf{R}}})$ in round i, abort regardless of whether $\sigma_{\mathsf{S}}^2 = \Delta_{\mathsf{R}} c_{\mathsf{S}}^i + K_{a_{\mathsf{R}}} b_{\mathsf{S}} + K_{r_{\mathsf{R}}}$.

$\mathcal{H}_{\{n+1,\ldots,2n\}}$**:** For $i = n+1, \ldots, 2n$, $\mathcal{H}_i$ is identical to $\mathcal{H}_{i-1}$, except for the following. If $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_6) \neq (\Delta_{\mathsf{S}}, a_{\mathsf{R}}, \Delta_{\mathsf{S}} a_{\mathsf{R}}, -K_{a_{\mathsf{S}}}, \Delta_{\mathsf{S}})$, set $\gamma_{\mathsf{S}}^{\mathsf{glo}} \leftarrow_{\$} \mathbb{F}_q$.

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

We first observe that for each round i, every value of the sender unknown to $\mathcal{A}_{\mathsf{R}}$ can be expressed in terms of the MAC key Δ_{R} and inputs $a_{\mathsf{S}}, b_{\mathsf{S}}$ and s_{S}, i.e. we have that

$$K_{a_{\mathsf{R}}} = M_{a_{\mathsf{R}}} - \Delta_{\mathsf{R}} a_{\mathsf{R}} \tag{19}$$

$$K_{r_{\mathsf{R}}} = M_{r_{\mathsf{R}}} - \Delta_{\mathsf{R}} r_{\mathsf{R}} \tag{20}$$

$$M_{a_{\mathsf{S}}} = \Delta_{\mathsf{S}} a_{\mathsf{S}} + K_{a_{\mathsf{S}}} \tag{21}$$

Simulator $\mathcal{S}_{\mathsf{R}}$

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

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

Authenticated value: Simulate according to Π_{Deal}.

Authenticated multiplication ($\mathcal{S}_{\mathsf{mul}}$): Simulate Π_{Deal} and let (α_1, β_1) be $\mathcal{A}_{\mathsf{R}}$'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 $(\alpha_1, \alpha_4) \neq (\Delta_{\mathsf{S}}, -K_{a_{\mathsf{S}}})$, set $\gamma_{\mathsf{S}}^1 \leftarrow_{\$} \mathbb{F}_q$.
- If $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_{\mathsf{R}}, M_{a_{\mathsf{R}}}, r_{\mathsf{R}}, M_{r_{\mathsf{R}}})$, abort.
- If $(\alpha_2, \alpha_3) \neq (a_{\mathsf{R}}, \Delta_{\mathsf{S}} a_{\mathsf{R}})$, set $\gamma_{\mathsf{S}}^2 \leftarrow_{\$} \mathbb{F}_q$.
- If $\alpha_6 \neq \Delta_{\mathsf{S}}$, set $\gamma_{\mathsf{S}}^3 \leftarrow_{\$} \mathbb{F}_q$.

Continue the simulation according to Π_{Deal} and compute $K_{c_{\mathsf{S}}} = K_{c_{\mathsf{S}}^l} + K_{c_{\mathsf{S}}^i} - K_{r_{\mathsf{S}}}$, if no abort occurs. Output $K_{c_{\mathsf{S}}}$.

Fig. 11. Simulator $\mathcal{S}_{\mathsf{R}}$ against a corrupted receiver.

$$M_{b_{\mathsf{S}}} = \Delta_{\mathsf{S}} b_{\mathsf{S}} + K_{b_{\mathsf{S}}} \tag{22}$$
$$M_{s_{\mathsf{S}}} = \Delta_{\mathsf{S}} s_{\mathsf{S}} + K_{s_{\mathsf{S}}} \tag{23}$$
$$M_{c_{\mathsf{S}}^l} = \alpha_1 a_{\mathsf{S}} b_{\mathsf{S}} + \beta_1 \tag{24}$$
$$M_{c_{\mathsf{S}}^i} = \alpha_3 b_{\mathsf{S}} + \beta_3 \tag{25}$$
$$c_{\mathsf{S}}^i = \alpha_2 b_{\mathsf{S}} + \beta_2 \tag{26}$$
$$d_{\mathsf{S}} = c_{\mathsf{S}}^i - s_{\mathsf{S}} = \alpha_2 b_{\mathsf{S}} + \beta_2 - s_{\mathsf{S}} \tag{27}$$
$$\sigma_{\mathsf{S}}^1 = \alpha_4 b_{\mathsf{S}} + \beta_4 \tag{28}$$
$$\sigma_{\mathsf{S}}^2 = \alpha_5 b_{\mathsf{S}} + \beta_5 \tag{29}$$
$$\sigma_{\mathsf{S}}^3 = \alpha_6 b_{\mathsf{S}} + \beta_6 \tag{30}$$

Thus, all of these values are uniformly distributed given $\mathcal{A}_{\mathsf{S}}$'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 $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_{\mathsf{R}}, M_{a_{\mathsf{R}}}, r_{\mathsf{R}}, M_{a_{\mathsf{R}}})$, then the sender will always abort in $\mathcal{H}_i$, possibly allowing $\mathcal{Z}$ to distinguish. We will now show that in case $(\alpha_1, \alpha_2, \beta_1, \beta_2) \neq (a_{\mathsf{R}}, M_{a_{\mathsf{R}}}, r_{\mathsf{R}}, M_{a_{\mathsf{R}}})$ 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 σ^2 as follows using Equations (19), (20), (26) and (29).

$$\begin{aligned}
0 &= \Delta_{\mathsf{R}} c_{\mathsf{S}}^i + K_{a_{\mathsf{R}}} b_{\mathsf{S}} + K_{r_{\mathsf{R}}} - \sigma_{\mathsf{S}}^2 \\
&= \Delta_{\mathsf{R}}(\alpha_2 b_{\mathsf{S}} + \beta_2) + K_{a_{\mathsf{R}}} b_{\mathsf{S}} + K_{r_{\mathsf{R}}} - \alpha_5 b_{\mathsf{S}} - \beta_5 \\
&= \Delta_{\mathsf{R}} \alpha_2 b_{\mathsf{S}} + \Delta_{\mathsf{R}} \beta_2 + (M_{a_{\mathsf{R}}} - \Delta_{\mathsf{R}} a_{\mathsf{R}}) b_{\mathsf{S}} + M_{r_{\mathsf{R}}} - \Delta_{\mathsf{R}} r_{\mathsf{R}} - \alpha_5 b_{\mathsf{S}} - \beta_5 \\
&= b_{\mathsf{S}}(M_{a_{\mathsf{R}}} - \alpha_5) + \beta_5 - M_{r_{\mathsf{R}}} + \Delta_{\mathsf{R}}((\alpha_2 - a_{\mathsf{R}}) b_{\mathsf{S}} + \beta_2 - r_{\mathsf{R}})
\end{aligned} \tag{31}$$

Define the random variables

$$\begin{aligned}
L &= b_{\mathsf{S}}(\alpha_5 - M_{a_{\mathsf{R}}}) + \beta_5 - M_{r_{\mathsf{R}}} \\
R &= b_{\mathsf{S}}(\alpha_2 - a_{\mathsf{R}}) + \beta_2 - r_{\mathsf{R}},
\end{aligned}$$

and we can simplify Equation (31) into

$$L = \Delta_{\mathsf{R}} \cdot R.$$

Assume first that $\alpha_2 = a_{\mathsf{R}}$ and $\beta_2 = r_{\mathsf{R}}$. In this case the random variable R is constant 0 and it must either hold that $\alpha_5 \neq M_{a_{\mathsf{R}}}$ or $\beta_5 \neq M_{r_{\mathsf{R}}}$. Therefore, the variable L is either constant non-zero or uniformly random over $\mathbb{F}_q$. Thus, the probability that Equation (31) holds is at most $1/|\mathbb{F}_q|$.

Now, assume that $\alpha_2 \neq a_{\mathsf{R}}$ or $\beta_2 \neq r_{\mathsf{R}}$. 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 (31) holds is at most $1/|\mathbb{F}_q|$ over the choice of Δ_{R}. Thus

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

We conclude that if $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_{\mathsf{R}}, M_{a_{\mathsf{R}}}, r_{\mathsf{R}}, M_{a_{\mathsf{R}}})$, 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 $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_6) \neq (\Delta_{\mathsf{S}}, a_{\mathsf{R}}, \Delta_{\mathsf{S}} a_{\mathsf{R}}, -K_{a_{\mathsf{S}}}, \Delta_{\mathsf{S}})$ and pass the check of γ^{glo}.

Towards this, we rewrite all checks for γ^1, γ^2 and γ^3 using the above identities. For γ^1, we apply Equations (21), (24) and (28) and get

$$\begin{aligned}0 &= b_{\mathsf{S}} M_{a_{\mathsf{S}}} + \sigma^1_{\mathsf{S}} - M_{c^l_{\mathsf{S}}} \\ &= b_{\mathsf{S}}(\Delta_{\mathsf{S}} a_{\mathsf{S}} + K_{a_{\mathsf{S}}}) + \alpha_4 b_{\mathsf{S}} + \beta_4 - (\alpha_1 a_{\mathsf{S}} b_{\mathsf{S}} + \beta_1) \\ &= b_{\mathsf{S}}((\Delta_{\mathsf{S}} - \alpha_1) a_{\mathsf{S}} + K_{a_{\mathsf{S}}} + \alpha_4) + \beta_4 - \beta_1 \end{aligned} \tag{32}$$

If $\alpha_1 = \Delta_{\mathsf{S}}$ but $\alpha_4 \neq -K_{a_{\mathsf{S}}}$, the term depending on b_{S} in Equation (32) does not vanish, hence the result will be uniformly distributed. The same holds for the case $\alpha_1 \neq \Delta_{\mathsf{S}}$ because of the dependence on the unknown a_{S}. Combined, if $(\alpha_1, \alpha_4) \neq (\Delta_{\mathsf{S}}, -K_{a_{\mathsf{S}}})$, the value γ^1 is uniformly distributed given $\mathcal{A}_{\mathsf{R}}$'s view.

Moving on, the check for γ^2 can be rewritten as

$$\begin{aligned}0 &= M_{c^i_{\mathsf{S}}} - M_{s_{\mathsf{S}}} - (\Delta_{\mathsf{S}} d_{\mathsf{S}} + K_{c^i_{\mathsf{S}}} - K_{s_{\mathsf{S}}}) \\ &= \alpha_3 b_{\mathsf{S}} + \beta_3 - (\Delta_{\mathsf{S}} s_{\mathsf{S}} + K_{s_{\mathsf{S}}}) - (\Delta_{\mathsf{S}}(\alpha_2 b_{\mathsf{S}} + \beta_2 - s_{\mathsf{S}}) + K_{c^i_{\mathsf{S}}} - K_{s_{\mathsf{S}}}) \\ &= b_{\mathsf{S}}(\alpha_3 - \Delta_{\mathsf{S}} \alpha_2) + \beta_3 - \Delta_{\mathsf{S}} \beta_2 - K_{c^i_{\mathsf{S}}} \end{aligned} \tag{33}$$

using the identities in Equations (23), (25) and (27). If $(\alpha_2, \alpha_3) \neq (a_{\mathsf{R}}, M_{a_{\mathsf{R}}})$, Equation (33) will only hold with probability $2/|\mathbb{F}_q|$. This follows from Equation (31) (i.e. it has to hold that $\alpha_2 = a_{\mathsf{R}}$), and the fact that Equation (33) is uniformly distributed if $\alpha_3 \neq \Delta_{\mathsf{S}} \alpha_2$.

Finally, we apply Equations (22) and (30) to the check for γ^3 and get

$$0 = M_{b_{\mathsf{S}}} - \sigma^3_{\mathsf{S}} - (K_{b_{\mathsf{S}}} - \sigma^3_{\mathsf{R}})$$

$$= \Delta_{\mathsf{S}} b_{\mathsf{S}} + K_{b_{\mathsf{S}}} - \alpha_6 b_{\mathsf{S}} - \beta_6 - K_{b_{\mathsf{S}}} + \sigma^3_{\mathsf{R}}$$
$$= b_{\mathsf{S}}(\Delta_{\mathsf{S}} - \alpha_6) + \sigma^3_{\mathsf{R}} - \beta_6 \tag{34}$$

Thus, if $\alpha_6 \neq \Delta_{\mathsf{S}}$, Equation (34) will be uniformly distributed. Combining all these observations, and considering the fact that $\gamma^{\mathsf{glo}} = \gamma^1 + \gamma^2 + \gamma^3$, it follows that the probability that $\mathcal{A}_{\mathsf{R}}$ produces a correct $\gamma^{\mathsf{glo}}_{\mathsf{R}}$ is upper bounded by $2/|\mathbb{F}_q|$, if $(\alpha_2, \alpha_5, \beta_2, \beta_5) \neq (a_{\mathsf{R}}, M_{a_{\mathsf{R}}}, r_{\mathsf{R}}, M_{a_{\mathsf{R}}})$. Thus we can bound the statistical distance between $\mathcal{H}_i$ and $\mathcal{H}_{i-1}$ by $2/|\mathbb{F}_q|$.

We conclude that the statistical distance between $\mathcal{H}_0$ and $\mathcal{H}_{2n}$ can be upper bounded by $4n/|\mathbb{F}_q|$, which is negligible in the security parameter. □

Efficiency For a complete set of authenticated triples, Π_{Deal} requires an 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 γ^{glo} is only done once over all triples and thus amortized away.

Since the protocol is symmetric for both parties, in summary 22 calls to $\mathbb{F}_q$-OLE are necessary for one authenticated triple. 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 $\mathcal{F}_{\mathsf{Deal}}$-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. Consider 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 $z = x + y, M_z = M_x + M_y = \Delta(x + y) + K_x + K_y$, 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.

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

Proof. **Correctness.** Correctness of the protocol is immediate with the exception of the multiplication step. Let $[a] = [a_{\mathsf{A}}|a_{\mathsf{B}}]$,$[b] = [b_{\mathsf{A}}|b_{\mathsf{B}}]$ and $[c] = [c_{\mathsf{A}}|c_{\mathsf{B}}]$ such that $c = a \cdot b$. Further let $\delta_{a_{\mathsf{A}}} = a_{\mathsf{A}} - x_{\mathsf{A}}$, $\delta_{b_{\mathsf{A}}} = b_{\mathsf{A}} - y_{\mathsf{A}}$, $\delta_{a_{\mathsf{B}}} = a_{\mathsf{B}} - x_{\mathsf{B}}$, $\delta_{b_{\mathsf{B}}} = b_{\mathsf{B}} - y_{\mathsf{B}}$, $\delta_x = \delta_{a_{\mathsf{A}}} + \delta_{a_{\mathsf{B}}}$ and $\delta_y = \delta_{b_{\mathsf{A}}} + \delta_{b_{\mathsf{B}}}$. Then

$$\begin{aligned}(a_{\mathsf{A}} + a_{\mathsf{B}}) \cdot (b_{\mathsf{A}} + b_{\mathsf{B}}) &= (x_{\mathsf{A}} + \delta_{a_{\mathsf{A}}} + x_{\mathsf{B}} + \delta_{a_{\mathsf{B}}}) \cdot (y_{\mathsf{A}} + \delta_{b_{\mathsf{A}}} + y_{\mathsf{B}} + \delta_{b_{\mathsf{B}}}) \\ &= (x_{\mathsf{A}} + x_{\mathsf{B}} + \delta_x) \cdot (y_{\mathsf{A}} + y_{\mathsf{B}} + \delta_y) \\ &= x \cdot y + y_{\mathsf{A}}\delta_x + y_{\mathsf{B}}\delta_x + x_{\mathsf{A}}\delta_y + x_{\mathsf{B}}\delta_y + \delta_x\delta_y\end{aligned}$$

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

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

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

Add: On input $(\texttt{add}, vid_1, vid_2, vid_3)$ 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 $(\texttt{mult}, vid_1, vid_2, vid_3)$ 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 $(\texttt{output}, \mathsf{P}, vid)$ from both parties, retrieve (vid, x) and output it to P.

Fig. 12. Ideal arithmetic two-party computation functionality.

Protocol Π_{2PC}

Init: A receives Δ_{B} from $\mathcal{F}_{\mathsf{Deal}}$, B receives Δ_{A}.

Rand: A and B ask $\mathcal{F}_{\mathsf{Deal}}$ for random authenticated values $[r_{\mathsf{A}}]_{\mathsf{A}}$,$[r_{\mathsf{B}}]_{\mathsf{B}}$ and store $[r] = [r_{\mathsf{A}}|r_{\mathsf{B}}]$.

Input: If $\mathsf{P} = \mathsf{A}$, then A asks $\mathcal{F}_{\mathsf{Deal}}$ for a random authenticated value $[x_{\mathsf{A}}]_{\mathsf{A}}$ and announces $x_{\mathsf{B}} =$ x - x_{A}. The parties build $[x_{\mathsf{B}}]_{\mathsf{B}}$ and define $[x] = [x_{\mathsf{A}}|x_{\mathsf{B}}]$ (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 $[z_{\mathsf{A}}]_{\mathsf{A}} = [x_{\mathsf{A}}]_{\mathsf{A}} + [y_{\mathsf{A}}]_{\mathsf{A}}$, symmetrically for B. This can be done locally.

Mult: A and B retrieve $[x]$, $[y]$ and ask $\mathcal{F}_{\mathsf{Deal}}$ for a random authenticated multiplication $[a] = [a_{\mathsf{A}}|a_{\mathsf{B}}]$,$[b] = [b_{\mathsf{A}}|b_{\mathsf{B}}]$ and $[c] = [c_{\mathsf{A}}|c_{\mathsf{B}}]$ such that $c = a \cdot b$. Additionally, A retrieves a random authenticated value $[r_{\mathsf{A}}]_{\mathsf{A}}$ from $\mathcal{F}_{\mathsf{Deal}}$. A and B compute $z = x \cdot y$ as follows:

1. A computes $\delta_{a_{\mathsf{A}}} = a_{\mathsf{A}} - x_{\mathsf{A}}$ and $\delta_{b_{\mathsf{A}}} = b_{\mathsf{A}} - y_{\mathsf{A}}$. Symmetrically, B computes $\delta_{a_{\mathsf{B}}} = a_{\mathsf{B}} - x_{\mathsf{B}}$ and $\delta_{b_{\mathsf{B}}} = b_{\mathsf{B}} - y_{\mathsf{B}}$.
2. A and B exchange all δ-values and compute $\delta_x = \delta_{a_{\mathsf{A}}} + \delta_{a_{\mathsf{B}}}$ and $\delta_y = \delta_{b_{\mathsf{A}}} + \delta_{b_{\mathsf{B}}}$.
3. A computes $\delta = \delta_x \cdot \delta_y$. A and B create $[\delta_{\mathsf{A}}]_{\mathsf{A}}$ via $[r_{\mathsf{A}}]_{\mathsf{A}}$.
4. A locally computes: $[z_{\mathsf{A}}]_{\mathsf{A}} = [c_{\mathsf{A}}]_{\mathsf{A}} - [x_{\mathsf{A}}]_{\mathsf{A}}\delta_y - [y_{\mathsf{A}}]_{\mathsf{A}}\delta_x - [\delta_{\mathsf{A}}]_{\mathsf{A}}$.
5. B locally computes: $[z_{\mathsf{B}}]_{\mathsf{B}} = [c_{\mathsf{B}}]_{\mathsf{B}} - [x_{\mathsf{B}}]_{\mathsf{B}}\delta_y - [y_{\mathsf{B}}]_{\mathsf{B}}\delta_x$.

Thus we have $[z] = [z_{\mathsf{A}}|z_{\mathsf{B}}]$.

Output: The parties retrieve $[x] = [x_{\mathsf{A}}|x_{\mathsf{B}}]$. If A is to learn x, then B reveals x_{B}. If B is to learn x, then A reveals x_{A}.

Fig. 13. Secure two-party computation in the $\mathcal{F}_{\mathsf{Deal}}$-hybrid model.

We thus can compute

$$z_{\mathsf{A}} = c_{\mathsf{A}} - y_{\mathsf{A}}\delta_x - x_{\mathsf{A}}\delta_y - \delta_x\delta_y$$
$$z_{\mathsf{B}} = c_{\mathsf{B}} - y_{\mathsf{B}}\delta_x - x_{\mathsf{B}}\delta_y$$

The value $\delta_x\delta_y$ has to be authenticated via a random authenticated value r_{A}: A sends $r' = \delta_x\delta_y - r_{\mathsf{A}}$ and $u = M_{r_{\mathsf{A}}} + t$ to B, where $t \leftarrow_{\$} \mathbb{F}_q$. Blocally computes $M'_{\delta_x\delta_y} = \Delta_{\mathsf{A}}r' + u$ 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.

Simulator $\mathcal{S}$

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

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

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

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

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

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

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 $\mathcal{F}_{\mathsf{Deal}}$.

Fig. 14. Simulator for Π_{2PC}.

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 [NNOB12].

Experiment 1 Sample a random global key $\Delta \in \mathbb{F}_q$. $\mathcal{A}_1$ is allowed to send queries $(\texttt{mac}, v, l)$, where $v \in \mathbb{F}_q$ and l is a fresh label. For each such query, sample a new local key $K \in \mathbb{F}_q$, store (l, K, v) and return $M = \Delta v + K$. If $\mathcal{A}_1$ outputs a query $(\texttt{break}, a_1, l_1, \ldots, a_p, l_p, v', M')$, 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 $(\texttt{mac}, v, l, M)$, 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 $(\texttt{break}, a_1, l_1, \ldots, a_p, l_p, v', M')$, 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 $(\texttt{mac}, v, l, M)$, 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 $(\texttt{break}, \Delta')$, 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 $(\texttt{break}, \Delta')$, and no `break`-query has been sent previously, if $\Delta' = \Delta$, $\mathcal{A}_4$ wins.

Lemma 3. *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 $(\texttt{mac}, v, l)$ from $\mathcal{A}_1$, $\mathcal{A}_2$ samples a random MAC $M \in \{0, 1\}_q^{\mathbb{F}}$, outputs $(\texttt{mac}, v, l, M)$ to Experiment 2 and sends M to $\mathcal{A}_1$. If $\mathcal{A}_1$ sends a query $(\texttt{break}, a_1, l_1, \ldots, a_p, l_p, v', M')$, $\mathcal{A}_2$ 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 4. *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 $(\texttt{mac}, v, l, M)$ 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 $(\texttt{break}, a_1, l_1, \ldots, a_p, l_p, v', M')$, 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 $(\texttt{break}, (M - M')/(v - v'))$ and wins Experiment 3. Obviously, $\mathcal{A}_2$ and $\mathcal{A}_3$ have the same success probability.

Lemma 5. *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.* [GNN17] based on noisy encodings seems to have the highest practical efficiency both communication-wise and computation-wise. From their paper, we can deduce the communication cost in field elements by

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

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 [GNN17] $\alpha = 8$ 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

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

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.

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.

Protocol	Security	Field	Comm./Triple
SPDZ [DPSZ12]	active	$\mathbb{F}_p$, 128-bit	430kB
	covert	$\mathbb{F}_p$, 128-bit	132kB
MASCOT [KOS16]	active	$\mathbb{F}_q$, 128-bit	360kB
	active	$\mathbb{F}_q$, 64-bit	106kB
This work	active	$\mathbb{F}_q$, 128-bit	22kB (11kB)
	active	$\mathbb{F}_q$, 64-bit	11kB (5.5kB)

Table 1. Comparison of communication overhead with existing solutions (taken from [KOS16]). Numbers in parentheses are obtained using optimistic parameters.

We currently have no implementation of the OLE protocol of [GNN17], 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

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.

[CvT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 110–123. Springer, Heidelberg, August 1995.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 445–465. Springer, Heidelberg, May 2010.

[DKM12] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Statistically secure linear-rate dimension extension for oblivious affine function evaluation. In Adam Smith, editor, *ICITS 12*, volume 7412 of *LNCS*, pages 111–128. Springer, Heidelberg, August 2012.

[DKMQ12] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. 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.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[FJN+13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 537–556. Springer, Heidelberg, May 2013.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *46th ACM STOC*, pages 495–504. ACM Press, May / June 2014.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GNN17] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. Cryptology ePrint Archive, Report 2017/409, 2017. http://eprint.iacr.org/2017/409.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 294–314. Springer, Heidelberg, March 2009.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *ACM CCS 16*, pages 830–842. ACM Press, 2016.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, Heidelberg, August 2013.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 476–494. Springer, Heidelberg, August 2014.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *31st ACM STOC*, pages 245–254. ACM Press, May 1999.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.