Commodity-Based 2PC for Arithmetic Circuits[⋆]

Ivan Damgård, Helene Haagh, Michael Nielsen, and Claudio Orlandi

Department of Computer Science, DIGIT, Aarhus University
{ivan,orlandi}@cs.au.dk

Abstract. We revisit the framework of Commodity-based Cryptography presented by Beaver (STOC'97) with a focus on updating the framework to fit with modern multiparty computation (MPC) protocols. We study the possibility of replacing the well-known preprocessing model with a commodity-based setting, where a set of *independent* servers (some of which may be corrupt) provide clients with correlated randomness. From this, the clients then distill correct and secure correlated randomness that they can use during the online phase of the MPC protocol. Beaver showed how to do OT with semi-honest security in the commodity setting. We improve on Beaver's result as follows: In a model where one of two clients and a constant fraction of the servers may be maliciously corrupted, we obtain unconditionally secure multiplication triples and oblivious linear evaluations (OLEs) such that the amortized communication cost of one triple/OLE is a constant number of field elements (when the field is sufficiently large). We also report on results from an implementation of the OLE protocol. Finally, we suggest an approach to practical realization of a commodity based system where servers need no memory and can be accessed asynchronously by clients, but still a maliciously corrupt client cannot get data he should not have access to.

Keywords. Secure Two-Party Computation, Information Theoretic Security, Oblivious Linear Evaluation, Commodity-based Cryptography.

1 Introduction

In commodity-based cryptography as defined in [Bea97], we have a set of clients (typically 2) and a set of servers. The clients want to use the servers to help them implement some cryptographic primitive in a way that is faster or more secure than if the clients were on their own – even if some of the servers and clients are corrupted. The primitive we focus on here is secure function evaluation, or primitives that are complete for this purpose, such as creation of random Beaver triples or oblivious linear evaluations (OLEs) (see more details on these below).

While the client-server model has been used multiple times to improve performances of MPC protocols (e.g. [BLW08,BCD+09,JNO14]), the commodity-based

[⋆] Research funded by: the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO), No 803096 (SPEC) and No 731583 (SODA); the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC);

model is significantly different. In particular, what sets the commodity-based model apart from other client-server models are the requirements made on the communication between servers and clients (we recall the formal definition in Section 2.2):

– Each server should be oblivious to the existence, identities and number of other servers, so no communication takes place between servers.
– The interaction between client i and server j should take the form of a 2-message protocol where the client sends a request $q_{i,j}$ and the server returns a response $r_{i,j}$.
– $q_{i,j}$ should be independent of the client's input (apart from its length) and of any previous communications with the servers.

The idea behind the commodity-based model is that it should be easy for a server to set up a business where it provides resources to anyone who is willing to pay, and on the other hand clients can decide to access as many servers as they see fit in order to gain confidence that at least some fraction of the material received is securely and correctly generated.

In [Bea97] it was shown how to do 2-party OT and hence 2-party secure computation in the commodity model, assuming *only passive corruption* of one client and a minority of the servers.[1] Note that in [Bea97] it is also claimed that the protocol can be modified to tolerate active corruptions, but no full protocol description nor security proof is provided.

In this paper, we revisit the commodity-based model and improve on Beaver's original result in several ways.[2] We present two protocols: one that produces batches of OLEs for two clients over any field. The protocol has statistical security against a bounded number of *maliciously* corrupted servers and 1 *maliciously* corrupted client. This improves on Beaver's result since: we can deal with arbitrary fields, the mechanism for dealing with active security is more efficient and (crucially) proven secure and, we modify the protocol so that it allows to produce *batches* of OLEs at the price of tolerating fewer corruptions, thus allowing for meaningful security-efficiency tradeoffs. Our second protocol produces multiplication triples over any field for two clients with statistical security against a bounded number of *maliciously* corrupted servers *or* 1 *maliciously* corrupted client. While the security guarantees of this protocol are weaker than the first one, the protocol is more efficient and therefore it could provide an interesting security-efficiency tradeoffs in some application. In Appendix A we illustrate how our constructions fit in the bigger picture of performing secure 2-party computation. Furthermore, we suggest an approach to practical realization of a commodity-based system where servers can be fully stateless and can be accessed asynchronously by clients.

[1] This corruption bound is clearly optimal for information theoretic security: if one of the two clients and half the servers could be corrupt, then we would immediately get a 2-party information theoretically secure OT which is well known to be impossible.

[2] In 2015, Tonicelli et al. [TND$^+$15] proposed a protocol which they claim to be in the commodity-based model. However, their construction assumes that the servers are trusted, as oppose to our constructions and the original paper by Beaver.

Constructing Commodity-based OLEs. Our first contribution (in Section 3) is a novel protocol that produces batches of m OLEs[3] over any field for two clients with statistical security against t *maliciously* corrupted servers and one client assuming that there are $n = 2t + 2m + 1$ servers. This protocol works in the commitment hybrid model. We also show how to do without commitments if we assume $n = 2t + 2m + 3$ servers. Thus, we can achieve essentially the optimal corruption threshold by setting the batch size to be $m = 1$, while if one is willing to assume a larger number of honest players it is possible to achieve higher amortized efficiency. When the underlying field is large (of size exponential in the security parameter), our protocol requires each server to supply just one OLE to the clients, and sending a constant number of field elements between the clients. This means that we can set both t and m to be $\Theta(n)$ and obtain an amortized communication cost per OLE corresponding to a constant number of field elements while still tolerating a constant fraction of corrupted servers. The computational work is dominated by polynomial interpolation, so by standard FFT techniques the computational complexity for $m \in \Theta(n)$ OLEs is $n \cdot \text{polylog}(n)$ elementary field operations.

We emphasize that the protocol between the clients and each individual server is independent of n, t and m. Thus the clients can collect data from any number of servers and later decide on the fly how many OLE's they think it will be secure to extract. Concretely, the clients may agree on number of servers n and corruption threshold t which determine the maximum m by our main theorem. Now, to obtain a total of M OLEs, the clients run $l = M/m$ parallel instances of the whole protocol, which is secure by parallel UC composition.

Although the goal of our protocol is OLE, we construct an equivalent but randomized version, called ROLE. This randomized primitive outputs to the clients random a, b as well as c_A and c_B where $ab = c_A + c_B$ (see Section 2.4). This is sufficient for 2-party computation with good concrete efficiency: ROLE trivially implies passively secure multiplication. But we can also use the ROLEs to produce authenticated multiplication triples allowing us to achieve malicious (and information theoretic) security.

In [DGN$^+$17], it was shown that actively secure 2-party computation can be built from actively secure OLE at an amortized price of 22 OLEs per multiplication. It was recently announced that this has been now improved to 14[4]. Using our protocol, this translates to asking for 14 ROLEs from each server per secure multiplication (Given our ROLE protocol, it is also possible to implement secure computation for more than two parties, by having each pair of parties constructing the necessary number of ROLEs with the help of the commodity servers). The construction from [DGN$^+$17] works by building authenticated multiplication triples from OLE. Such a triple consists of random values x, y, z with $xy = z$ as well as additive sharings among Alice and Bob of the three values. Moreover,

[3] The OLE (Oblivious Linear Evaluation) primitive is defined as follows: Alice inputs values a and b in some field $\mathbb{F}$ and Bob inputs $x \in \mathbb{F}$. Alice learns nothing and Bob learns $y = ax + b$.

[4] Personal communication.

the shares are authenticated using an unconditionally secure MAC scheme to prevent cheating.

In more details, our protocol for ROLE works as follows: the clients ask each server for a ROLE $(a_i, b_i, c_{A,i}, c_{B,i})$, where Alice holds $(a_i, c_{A,i})$ and Bob holds $(b_i, c_{B,i})$. Alice and Bob then create two polynomials $A(X)$ and $B(X)$ of degree d by using some of the a_i's and b_i's. Then they jointly create two polynomials $C_A(X)$ and $C_B(X)$ of degree $2d$ by using some of the $c_{A,i}$'s and $c_{B,i}$'s, and to get the rest of the points they use the remaining unused ROLES from the servers to multiply $A(X)$ and $B(X)$. If every party behaved honestly, then Alice and Bob will now hold four polynomials that satisfy the following equation:

$$A(x) \cdot B(x) = C_A(x) + C_B(x) \quad \text{for all } x \in \mathbb{F}.$$

However, if one of the servers provided a bad ROLE or one of the clients behaved inconsistently during the protocol, then some of the polynomials might not be well defined, and hence the input/output is not be well defined either. To be able to detect cheating, we introduce a check phase, where each client chooses a random challenge (a random field element), where the other client must prove that the equation holds when evaluating the polynomials on the challenge value. The non-trivial part of our proof is to show that just one such check in each direction is sufficient to ensure that well-defined polynomials exist for a corrupt client (whenever the protocol does not abort). If the protocol does not abort, then the clients can output a valid ROLE by computing $A(\sigma) \cdot B(\sigma) = C_A(\sigma) + C_B(\sigma)$ where σ is a predefined and unused field element. The check for dealing with active adversaries, as well as its analysis, are the main changes we introduce to the original protocol by Beaver, which was only proved to be passive secure.

Implementation of Commodity-based OLEs. We experimentally validate our OLE protocol in Section 4, by implementing it and testing it on Amazon Cloud. The servers were spread over different locations (5 different continents). We give detailed timings for different settings of parameters. As an example, when we want to tolerate 5 corrupt servers and hence involve 13 servers in total, the latency for generating an OLE is about 1 msec. The amortized time for one OLE ranges from 0.02 msec to 0.5 msec when the field size in bits ranges from 32 to 2048. Note that a growth in wall clock time when the field size grows is to be expected: although the number of field elements to send is constant in the field size, we need to send and process more bits when the field size is larger. These timings are for the case where each instance of the protocol runs with $m = 1$, so we can increase m and decrease t and get better amortized times. For instance, if we are willing to assume at most 1 corrupted server out of 13, one instance of the same protocol can produce batches of 5 OLEs at the time and the amortized times instead vary between 0.004 and 0.1 msec.

With the known reductions from multiplication triples to OLE, our performance numbers show that we can preprocess a secure multiplication triple in amortized time between 0.08 and 0.4 msec for a field size of 128 bits, depending on the assumed corruption threshold. For a very rough comparison, the recent "Overdrive" protocol [KPR18], which does the same task using only communication between the two clients, takes roughly 0.03 msec per multiplication in a LAN setting. We emphasize that it does not makes sense to compare the timings directly because the hardware set-ups are different and the achieved types of security are different: computational security in the "Overdrive" protocol versus unconditional security assuming an honest server majority in our protocol. Nevertheless, we believe the numbers show that our protocol is indeed applicable in a practical setting.

Constructing Commodity-based Multiplication Triples. As an additional contribution (in Section 5) we present a novel protocol for constructing multiplication triples directly (instead of producing OLE first and then using known reductions). For malicious (and information theoretic) security the standard goal is to produce authenticated multiplication triples. So it may seem natural to ask the servers for such triples and try to extract a secure triple from what we get. However, this does not work: the authentication in a triple involves unconditionally secure MACs and a server will of course know the MAC keys involved in its own triples. If this server as well as, say, Alice is corrupt, then Alice can cheat with those MACs. This problem can be solved in a weaker corruption model where *either* some servers *or* a client can be corrupt (but not both).

The high-level idea of the triple protocol closely follows the structure of the OLE protocol with some variation. The clients obtain random triples authenticated under different global keys from the servers, which means that the triples cannot be combined. To solve this, we observe that the MACs are key-homomorphic, which allows the clients to adjust the global MAC key with one round of interaction. Once the MACs are adjusted to use the same key, the clients can combine the triples in a similar manner as the OLE protocol.

Allowing Servers to be Memoryless. As a final contribution, we suggest an approach for the practical realization of a commodity based system. For the sake of presentation, we assume in most of the paper that there are secure channels between each pair of client and server. Also, we allow servers to remember which clients they talked to and what was sent (so that output to different clients can be properly correlated). Those assumptions are inconvenient in practice and therefore, in Section 6, we show how to get rid of both requirements. It will clearly be an advantage if the servers do not need memory in the sense that they do not have to remember who they talked to or what was sent. This would mean that a server will not have to administrate identities and log-in credentials, and could simply generate the required data on the fly, say, against payment in some cryptocurrency.

An obvious problem with this goal is that the data sent to two clients Alice and Bob must be correlated and a memoryless server cannot make this happen on its own. We solve this by letting Alice and Bob interact before talking to the severs, in order to choose a common nonce n_{AB}. Then Alice can send ("Alice", n_{AB}) to the server who will use this and a private PRF key to generate Alice's data. We can do something similar for Bob. However, this is not secure: if Alice is corrupt, she can send both ("Alice", n_{AB}) and ("Bob", n_{AB}) to each honest server, get Bob's data and break the protocol. We show how this can be solved by generating the nonce such that Alice and Bob each know different secrets relating to the nonce and hence cannot impersonate the other party towards the server.

Alternatives to the Commodity Model. The commodity model has not received much attention since its introduction, but a very large amount of work has been done on secure computation in general. So we should of course ask ourselves, if some of this work has made the model redundant. Now, if two clients want to do secure computation, two obvious alternatives are well known that allow the clients to do it themselves: first, one could use garbled circuits. This will be constant round, but introduces an overhead on communication that depends on security parameter and also grows with the size of the underlying field (if the goal is arithmetic computation). A second alternative is to do GMW style arithmetic computation using authenticated multiplication triples that the clients generate themselves. If the underlying field is not too large, the triples can be built very efficiently using OT extension [KOS16], but we get an overhead that grows linearly with the bit-size of a field element. Another recent approach generates OLEs from noisy encoding-style assumptions with constant overhead [ADI$^+$17], but only with passive security. For specific applications that use scalar-vector multiplication with long vectors, one may consider the generation of vector-OLE[5] as in [ADI$^+$17] obtaining rate 1/3. This primitive was recently improved to rate 1/2 using compression [BCGI18], that utilize function secret-sharing to enable a small "sparse" vector-OLE to be locally expanded to larger width.

Since it is clear that any solution involving only the two clients must be based on some intractability assumption, such solutions are incomparable to the commodity model with respect to security: we are replacing trust in a computational assumption by trust in some fraction of the servers. But even if we ignore this issue, what we can do in the commodity model seems to be competitive because our protocol has active security and the overhead of doing a secure multiplication is constant as the field size grows. We do not know any solution with these properties that the clients could execute themselves.

The commodity model is somewhat related to the idea of *combiners*: for instance, an Oblivious Transfer (OT) combiner [HIKN08,HKN$^+$05] is a protocol that gets (black-box) access to a number of OT implementations, some of which may be faulty, and the goal is now is to build a secure OT. One may think of the given OT implementations as corresponding to the servers in the commodity model. However, the models are incomparable: on one hand, the combiner model is more restrictive since its "servers" are assumed to only implement a certain primitive. On the other hand, a combiner may make many correlated calls to the "servers".

[5] In the width-w vector-OLE primitive, Alice inputs $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^w$ and Bob $x \in \mathbb{F}$. Alice learns nothing and Bob learns $\boldsymbol{y} = \boldsymbol{a}x + \boldsymbol{b}$

2 Preliminaries

Let $[n]$ be the set of integers $\{1, \ldots, n\}$. Denote a field of size q as $\mathbb{F}_q$ and the set of all polynomials over such field as $\mathbb{F}_q[X]$. All variables and operations are over $\mathbb{F}_q$ unless stated otherwise. $\boldsymbol{v}$ denotes a vector with entries in $\mathbb{F}_q$, the entries are usually denoted v_1, v_2... If $\boldsymbol{u}, \boldsymbol{v}$ are vectors of the same length, say m, then $\boldsymbol{v} * \boldsymbol{u}$ denotes the vector containing the coordinate-wise products, $\boldsymbol{v} * \boldsymbol{u} = (v_1u_1, v_2u_2, ..., v_mu_m)$. As a shorthand, we denote $\mathbb{F}$ to be a field of arbitrary size.

2.1 Security model

We will use the UC framework of Canetti [Can01] to prove our protocols secure. Informally, we compare a real protocol π between the parties to a setting in which the parties only talk with an ideal functionality $\mathcal{F}$, which by definition is secure. To model the information the adversary learns during the protocol execution, each ideal functionality is given a leakage port on which it leaks all the information the adversary is allowed to learn. Furthermore, to model the adversary's influence over the protocol and the corrupt players, each ideal functionality is given a influence port on which it can receive messages from the adversary. To prove the real protocol secure, we construct a simulator $\mathcal{S}$ such that no adversary controlling all malicious players can make an environment distinguish between the real protocol execution and the simulator's transcript. Intuitively, the adversary gains nothing from controlling the malicious players, that he could not have simulated himself offline. In particular we use the variant in which the environment plays the role of the adversary and will prove our protocols secure under static corruption for malicious adversaries.

2.2 Commodity Model

Commodity-based cryptography works in a client-server model, where a group of clients obtain some information (called commodities) from a set of servers. In the basic setting, the clients will send a request to a server, who will reply with a single response computed from the request, whereas other settings may extend this to multiple rounds.

Following the work of Beaver, we define a two-tiered (c, n)-protocol $\pi = (\mathcal{C}, \mathcal{S})$ as a collection of $c + n$ probabilistic interactive Turing machines (PTM), which are divided into two groups: the clients $\mathcal{C}$ and the servers $\mathcal{S}$. The clients must be polynomial time PTM's and are assigned a unique id $i \in [c]$, and the servers are likewise assigned a unique id $j \in [n]$. We consider the basic form of two clients $c = 2$ and a variable number of servers n, with the servers being polynomial time PTM's. Other settings, which we shall not consider, may include a variable number of clients, computationally unbounded servers, multiple rounds between clients and servers or even allowing communication between servers.

Definition 1 (Stateless Oblivious RPC, [Bea97]). *Given a two-pass protocol between client $C_i \in \mathcal{C}$ and server $S_j \in \mathcal{S}$, where the C_i sends a request $q_{i,j}$ to the S_j, who send back the response $r_{i,j}$. This protocol is called a* stateless oblivious remote procedure call (RPC) *if $q_{i,j}$ is independent of C_i's input x_i (apart from the length) and of any previous communications with S_j or any other servers (apart from including tags for identifying and authenticating C_i and S_j).*

Definition 2 (Commodity-based Protocol, [Bea97]). *A two-tiered protocol π is* commodity-based *if*
1. *No communication between servers is necessary.*
2. *Servers do not need to know the identities, numbers of, or existence of other servers.*
3. *For each client $C_i \in \mathcal{C}$ and server $S_j \in \mathcal{S}$, C_i interacts with S_j only through stateless oblivious RPC's.*

2.3 Commitments

Some of our protocol make use of a UC-secure commitment scheme, which is modelled by an ideal functionality $\mathcal{F}_{\text{COMMIT}}$. Committing to a value x is denoted $\text{COMMIT}(x)$ and means that the committer sends x to the commit functionality which notifes the other party that the commitment has been made. Opening is denoted $\text{OPEN}(x)$ and means the committer sends an open command to the functionality which then sends x to the other party. In a practical implementation, $\mathcal{F}_{\text{COMMIT}}$ can be implemented using a random oracle (it is well known, and trivial to prove, that applying a random oracle to the string to commit to, concatenated by random coins, gives a UC-secure commitment scheme).

2.4 Oblivious Linear Evaluation

An oblivious linear evaluation (OLE) over the finite field $\mathbb{F}$ is a primitive in which Alice inputs $a, b \in \mathbb{F}$ and Bob inputs $x \in \mathbb{F}$. Alice learns nothing and Bob learns $y = ax + b$. This primitive can be seen as a natural generalization of Oblivious Transfer (OT) [Rab05] for the case $\mathbb{F} = \mathbb{F}_2$ or as a special case of Oblivious Polynomial Evaluation (OPE) [NP99] for the case of degree 1 polynomials. The ideal UC-functionality $\mathcal{F}^m_{\text{OLE}}$ is defined in Figure 1. It implements m OLEs in parallel.

A variant called random oblivious linear evaluation (ROLE) is a similar primitive, but where Alice receives random values $u, w_1 \overset{\$}{\leftarrow} \mathbb{F}$ and Bob $v, w_2 \overset{\$}{\leftarrow} \mathbb{F}$ such that $uv = w_1 + w_2$. The ideal UC-functionality $\mathcal{F}^m_{\text{ROLE}}$ is defined in Figure 2. We show that a random oblivious linear evaluation (ROLE) is equivalent to an oblivious linear evaluation (OLE) in the same way oblivious transfer is show equal to a random oblivious transfer. We define a protocol π^m_{OLE} that realizes $\mathcal{F}^m_{\text{OLE}}$ with access to $\mathcal{F}^m_{\text{ROLE}}$. This protocol (which is folklore) is formally given in Figure 3 and can be proven secure as stated in the following:

Functionality $\mathcal{F}^m_{\text{OLE}}$

1. Upon receiving message $(\mathtt{inputA}, a, b)$ from Alice with $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^m$: if there already is a stored tuple from Alice, then ignore the message. Otherwise, store $\boldsymbol{a}$ and $\boldsymbol{b}$ and send message $(\mathtt{inputA})$ on leakage port.
2. Upon receiving message $(\mathtt{inputB}, \boldsymbol{x})$ from Bob with $\boldsymbol{x} \in \mathbb{F}^m$: if there already is a stored tuple from Bob, then ignore the message. Otherwise, store $\boldsymbol{x}$ and send message $(\mathtt{inputB})$ on leakage port.
3. Upon receiving message $(\mathtt{deliver}, \mathtt{A})$ on influence port: if $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{x}$ have been stored, then send $(\mathtt{delivered})$ to Alice. Otherwise, ignore the message.
4. Upon receiving message $(\mathtt{deliver}, \mathtt{B})$ on influence port: if $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{x}$ have been stored, then set $\boldsymbol{y} = \boldsymbol{a} * \boldsymbol{x} + \boldsymbol{b}$ and send $(\mathtt{output}, \boldsymbol{y})$ to Bob. Otherwise, ignore the message.

Fig. 1. Ideal functionality $\mathcal{F}^m_{\text{OLE}}$ for Oblivious Linear Evaluation (OLE).

Functionality $\mathcal{F}^m_{\text{ROLE}}$

1. Upon receiving message $(\mathtt{corrupt}, \mathtt{A}, (\boldsymbol{u}, \boldsymbol{w}_1))$ on influence port with $\boldsymbol{u}, \boldsymbol{w}_1 \in \mathbb{F}^m$: if no values for $\boldsymbol{u}, \boldsymbol{w}_1$ have been stored, draw and store uniformly random $\boldsymbol{v} \in \mathbb{F}^m$ and compute and store $\boldsymbol{w}_2 := \boldsymbol{u} * \boldsymbol{v} - \boldsymbol{w}_1$.
2. Upon receiving message $(\mathtt{corrupt}, \mathtt{B}, (\boldsymbol{v}, \boldsymbol{w}_2))$ on influence port with $\boldsymbol{v}, \boldsymbol{w}_2 \in \mathbb{F}^m$: if no values for $\boldsymbol{v}, \boldsymbol{w}_2$ have been stored, draw and store uniformly random $\boldsymbol{u} \in \mathbb{F}^m$ and compute and store $\boldsymbol{w}_1 := \boldsymbol{u} * \boldsymbol{v} - \boldsymbol{w}_2$.
3. Upon receiving message $(\mathtt{deliver}, \mathtt{A})$ on influence port: if no values for $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}_1$ and $\boldsymbol{w}_2$ have been stored, draw and store uniformly random $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}_1 \in \mathbb{F}^m$ and compute and store $\boldsymbol{w}_2 := \boldsymbol{u} * \boldsymbol{v} - \boldsymbol{w}_1$. Send $(\mathtt{output}, (\boldsymbol{u}, \boldsymbol{w}_1))$ to Alice.
4. Upon receiving message $(\mathtt{deliver}, \mathtt{B})$ on influence port: if no values for $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}_1$ and $\boldsymbol{w}_2$ have been stored, draw and store uniformly random $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}_1 \in \mathbb{F}^m$ and compute and store $\boldsymbol{w}_2 := \boldsymbol{u} * \boldsymbol{v} - \boldsymbol{w}_1$. Send $(\mathtt{output}, (\boldsymbol{v}, \boldsymbol{w}_2))$ to Bob.

Fig. 2. Ideal functionality $\mathcal{F}^m_{\text{ROLE}}$ for Random Oblivious Linear Evaluation (ROLE). It chooses random outputs for the parties, but lets a corrupt party choose his own outputs.

Lemma 1. *The protocol π^m_{OLE} UC-realizes $\mathcal{F}^m_{\text{OLE}}$ in the $\mathcal{F}^m_{\text{ROLE}}$-hybrid model.*

Correctness of the protocol can be trivially checked. Security can be proven similarly to other protocols in the correlated randomness model [IKM+13]: if Bob is corrupted, the simulator emulates the ROLE by picking random $\boldsymbol{v}, \boldsymbol{w}_2$, extracts $\boldsymbol{x}$ from $\boldsymbol{e}$ and $\boldsymbol{v}$ and feeds it to the ideal functionality to receive $\boldsymbol{y}$. Finally the simulator chooses a random $\boldsymbol{s}$ and computes $\boldsymbol{d} = \boldsymbol{v}^{-1} * (\boldsymbol{y} - \boldsymbol{w}_2 - \boldsymbol{s})$, thus producing a view which is distributed identically as in the real world. Here, $\boldsymbol{v}^{-1}$ means the vector with entries $v_1^{-1}, v_2^{-1}, \ldots$ In the case where Alice is corrupted, the simulator emulates the ROLE by picking random $\boldsymbol{u}, \boldsymbol{w}_1$ and sends a random $\boldsymbol{e}$. Then, upon receiving $\boldsymbol{d}$ and $\boldsymbol{s}$ the simulator extracts $\boldsymbol{a} = \boldsymbol{d} + \boldsymbol{u}$ and $\boldsymbol{b} = \boldsymbol{s} - \boldsymbol{w}_1 - \boldsymbol{a} * \boldsymbol{e}$ and feeds them to the ideal functionality.

Protocol π^m_{OLE}

Input: Alice inputs $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^m$ and Bob inputs $\boldsymbol{x} \in \mathbb{F}^m$
Output: Bob outputs $\boldsymbol{y}$ such that $\boldsymbol{y} = \boldsymbol{a} * \boldsymbol{x} + \boldsymbol{b}$

Protocol:
1. Both run $\mathcal{F}^m_{\text{ROLE}}$ such that Alice gets $\boldsymbol{u}, \boldsymbol{w}_1$ and Bob $\boldsymbol{v}, \boldsymbol{w}_2$
2. Bob computes and sends $\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{v}$;
3. Alice computes and sends $\boldsymbol{d} = \boldsymbol{a} - \boldsymbol{u}$ and $\boldsymbol{s} = \boldsymbol{w}_1 + \boldsymbol{a} * \boldsymbol{e} + \boldsymbol{b}$;
4. Bob returns $\boldsymbol{y} = \boldsymbol{w}_2 + \boldsymbol{x} * \boldsymbol{d} - \boldsymbol{d} * \boldsymbol{e} + \boldsymbol{s}$.

Fig. 3. Protocol for OLE in the $\mathcal{F}^m_{\text{ROLE}}$-hybrid model.

3 Commodity-based Oblivious Linear Evaluation

In this section, we present a commodity-based protocol for ROLE, which will combine commodities in the form of ROLE's from n servers into a secure ROLE (see Section 2.4 for the formal definition). We consider the setting where one client and up to t of servers are maliciously corrupt. We first present a protocol for the commitment-hybrid model and then show as a corollary a protocol that does not use commitments, at the expense of slightly worse parameters.

The high-level idea of the protocol is to first obtain the ROLE commodities from each server. The clients will use $d + 1$ of these to fully define two degree d polynomials $A(X)$ and $B(X)$ held by each party respectively and define points on two degree $2d$ polynomials $C_1(X)$ and $C_2(X)$, which shall form a secret sharing of $A(X) \cdot B(X)$. The clients will use the remaining ROLEs to securely compute $A(X) \cdot B(X)$ on agreed upon points until we have obtained enough points to fully define $C_1(X)$ and $C_2(X)$. After constructing these polynomials, the clients perform a check that with high probability reveals whether the polynomials are valid, that is the equation $A(\gamma) \cdot B(\gamma) = C_1(\gamma) + C_2(\gamma)$ holds for all $\gamma \in \mathbb{F}$. The resulting ROLEs of the protocol will be defined by a fixed set of points

$\sigma_1, ..., \sigma_m \in \mathbb{F}$ on these polynomials:

$$\begin{aligned} \boldsymbol{u} &= (A(\sigma_1), ..., A(\sigma_m)) & \boldsymbol{v} &= (B(\sigma_1), ..., B(\sigma_m)) \\ \boldsymbol{w}_1 &= (C_1(\sigma_1), ..., C_1(\sigma_m)) & \boldsymbol{w}_2 &= (C_2(\sigma_1), ..., C_2(\sigma_m)) \end{aligned}$$

Here, one client will output $\boldsymbol{u}, \boldsymbol{w}_1 \in \mathbb{F}^m$ and the other $\boldsymbol{v}, \boldsymbol{w}_2 \in \mathbb{F}^m$ such that $\boldsymbol{u} * \boldsymbol{v} = \boldsymbol{w}_1 + \boldsymbol{w}_2$. The complete protocol π_{ROLE} is presented in Protocol 1.

Protocol π^m_{ROLE}

Public: Let $d = t + m$ and $n = 2d + 1$ and let $\mathcal{U} = \{\gamma_1, \ldots, \gamma_n\}$, $\mathcal{V} = \{\sigma_1, ..., \sigma_m\}$ be sets of publicly known distinct points in $\mathbb{F}$. Let $\{\mathtt{S}_1, \ldots, \mathtt{S}_n\}$ be the servers and {Alice, Bob} the clients.
Output: Alice gets $\boldsymbol{u}, \boldsymbol{w}_1 \in \mathbb{F}^m$ and Bob get $\boldsymbol{v}, \boldsymbol{w}_2 \in \mathbb{F}^m$ such that $\boldsymbol{u} * \boldsymbol{v} = \boldsymbol{w}_1 + \boldsymbol{w}_2$.

Protocol:

```
Stateless oblivious RPC:
 1. Alice and Bob: send a request to each server $\mathtt{S}_i$ for $i \in [n]$;
 2. Server $\mathtt{S}_i$ for $i \in [n]$:
      – creates a ROLE $(a_i, b_i, c_{1,i}, c_{2,i}) \xleftarrow{\$} \mathbb{F}^4$ such that $a_i \cdot b_i = c_{1,i} + c_{2,i}$;
      – sends $(a_i, c_{1,i})$ to Alice and sends $(b_i, c_{2,i})$ to Bob;

Computation phase:
 3. Alice:
      – interpolates polynomial $A$ of degree $d$ by setting $A(\gamma_i) = a_i$ for $i \in [d+1]$;
      – prepares polynomial $C_1$ of degree $2d$ by setting $C_1(\gamma_i) = c_{1,i}$ for $i \in [d+1]$;
 4. Bob:
      – interpolates polynomial $B$ of degree $d$ by setting $B(\gamma_i) = b_i$ for $i \in [d+1]$;
      – prepares polynomial $C_2$ of degree $2d$ by setting $C_2(\gamma_i) = c_{2,i}$ for $i \in [d+1]$;
 5. For $i = d + 2, \ldots, n$
      a. Alice: sends $k_i = A(\gamma_i) - a_i$ to Bob;
      b. Bob:
           • sends $\ell_i = B(\gamma_i) - b_i$ to Alice;
           • sets $C_2(\gamma_i) = c_{2,i} + B(\gamma_i) \cdot k_i$;
      c. Alice: sets $C_1(\gamma_i) = c_{1,i} + A(\gamma_i) \cdot \ell_i - k_i \cdot \ell_i$;
 6. Alice: interpolates polynomial $C_1$ by the $2d + 1$ defined points;
 7. Bob: interpolates polynomial $C_2$ by the $2d + 1$ defined points;

Check phase:
 8. Alice: draw $t_A \xleftarrow{\$} \mathbb{F} \backslash (\mathcal{U} \cup \mathcal{V})$ uniformly at random and send $t_A$ to Bob;
 9. Bob:
      – draw $t_B \xleftarrow{\$} \mathbb{F} \backslash (\mathcal{U} \cup \mathcal{V})$ uniformly at random;
      – Execute $\textsc{commit}(B(t_A), C_2(t_A))$ and send $t_B$ to Alice;
10. Alice: once commitment has been done, send $(A(t_B), C_1(t_B))$ to Bob;
11. Bob:
      – check if $A(t_B) \cdot B(t_B) = C_1(t_B) + C_2(t_B)$ and abort if not;
      – execute $\textsc{open}(B(t_A), C_2(t_A))$;
```

12. Alice:
    - receive $(B(t_A), C_2(t_A))$ from the opening of the commitment, abort if nothing is received;
    - check if $A(t_A) \cdot B(t_A) = C_1(t_A) + C_2(t_A)$ and abort if not;

Output phase:

13. Alice: output $\boldsymbol{u} = (A(\sigma_1), ..., A(\sigma_m))$ and $\boldsymbol{w}_1 = (C_1(\sigma_1), ..., C_1(\sigma_m))$;
14. Bob: output $\boldsymbol{v} = (B(\sigma_1), ..., B(\sigma_m))$ and $\boldsymbol{w}_2 = (C_2(\sigma_1), ..., C_2(\sigma_m))$.

Protocol 1: Protocol for commodity-based ROLE

Theorem 1. *Assume that $n = 2t + 2m + 1$ and that $|\mathbb{F}|$ is exponential in the security parameter. Then protocol π^m_{ROLE} is an implementation of $\mathcal{F}^m_{\text{ROLE}}$ in the $\mathcal{F}_{\text{COMMIT}}$-hybrid model with statistical UC-security. The protocol tolerates a static adversary corrupting one client and at most $t \leq \frac{n-2m-1}{2}$ servers. The simulation is perfect unless an error event occurs, which has probability at most $\frac{n}{|\mathbb{F}|-(n+m)}$.*

Proof. The environment corrupts one client and a subset of the servers $\mathcal{C} \subset [n]$ with $|\mathcal{C}| \leq t$. Thus, the environment learns at most t points on all polynomials, and one extra from the check phase. After seeing these points, the environment still cannot distinguish whether it is interacting with the ideal world or the real world – that is, the output of the computation (which is m extra points on the polynomial) is compatible with the points that it already received from the simulator (which does not learn the output of the honest party). In other words, a necessary condition for being able to prove unconditional security is to set the degree to $d = t + m$ (since a random such polynomial has $t + m + 1$ random coefficients). The protocol uses $2d + 1$ ROLEs in total and thus we need the number of servers to be at least $n \geq 2d + 1 = 2t + 2m + 1$. In other words, we have that $t \leq \frac{n-2m-1}{2}$.

The protocol is essentially symmetric, i.e. the only difference is that in the proof for corrupt Bob we need to exploit that Bob has to commit to his answer to Alice's challenge before he sees $A(t_B)$, and hence a corrupt Bob also knows only t points of Alice's polynomial. Thus, in the following we prove security against corrupt Alice.

Let $\mathcal{C} \subset [n]$ denote the set of corrupt servers and $\mathcal{H} \subseteq [n]$ the set of honest servers. For these sets it hold that $|\mathcal{C}| \leq t$, $\mathcal{C} \cup \mathcal{H} = [n]$, and $\mathcal{C} \cap \mathcal{H} = 0$. Note that the servers play two different roles in the protocol: the first $d + 1$ servers are used to define the polynomials $A(x)$ and $B(x)$, and the last $n - d - 1$ servers are used to compute the remaining points on $C_1(x)$ and $C_2(x)$.

We start by presenting some facts about the protocol.

Definition 3. *For an honest server $i \in \mathcal{H}$, Alice gets a_i from the server and (if $i \geq d + 2$) sends a value k_i to Bob. Note that if Alice follows the protocol, then the values $\{a_i\}_{i \in \mathcal{H}, i \leq d+1}$ and $\{k_i + a_i\}_{i \in \mathcal{H}, i \geq d+2}$ are all consistent with a polynomial of degree at most d (namely $A(x)$ if Alice was honest). We say that Alice* behaves consistently *if such a polynomial $A^*(x)$ of degree at most d exists.*

Lemma 2. *If Alice is corrupt, but behaves consistently, then from her interaction with honest servers and Bob, one can compute a uniquely defined view for Alice and all corrupt servers that is consistent with them having followed the protocol up to (but not including) the check phase. This includes polynomials $A^*(x), C_1^*(x)$ of degree at most d and $2d$ respectively, where if Alice is honest we have $A^*(x) = A(x)$ and $C_1^*(x) = C_1(x)$.*

Proof. Since Alice behaves consistently this uniquely defines a polynomial $A^*(x)$ of degree at most d, by the above definition. Now we define a view $(a_i, b_i, c_{1,i}, c_{2,i})$ for each corrupt server $i \in \mathcal{C}$: The server sent $b_i, c_{2,i}$ to honest Bob, so these are fixed. Then if $1 \leq i \leq d+1$, there is no interaction between Alice and Bob so we set $a_i = A^*(\gamma_i)$ and $c_{1,i} = a_i b_i - c_{2-i}$. If $d+2 \leq i \leq n$, Alice sent k_i and received ℓ_i from Bob. So we set $a_i = A^*(\gamma_i) - k_i$, and $c_{1,i} = a_i b_i - c_{2-i}$. This gives a view for each corrupt server which is consistent with $A^*(x)$ and honest behaviour. Each honest server i, has sent $a_i, c_{1,i}$ to Alice and (if $i \geq d+2$) she has received a value ℓ_i from Bob and has sent k_i to Bob.

In particular, $c_{1,i}$ is now defined for all i and k_i, ℓ_i are defined for $i \geq d+2$. Therefore we can define the polynomial $C_1^*(x)$ by simply following the specification of the protocol, namely we set $C_1^*(\gamma_i) = c_{1,i}$ for $1 \leq i \leq d+1$ and otherwise $C_1^*(\gamma_i) = c_{1,i} + A^*(\gamma_i)\ell_i - k_i \ell_i$, and finally we interpolate $C_1^*(x)$ from these values. □

Lemma 3. *If Alice is corrupt and does not behave consistently, then the protocol aborts in the check phase except with probability at most $\frac{n}{|\mathbb{F}|-(m+n)}$.*

Proof. Consider the values $Q = \{a_i\}_{i \in \mathcal{H}, i \leq d+1} \cup \{k_i + a_i\}_{i \in \mathcal{H}, i \geq d+2}$ (see Definition 3). Since Alice does not behave consistently these values are not all consistent with a polynomial of degree at most d. We nevertheless define a polynomial $A'(x)$ by interpolating from the first $d+1$ values in Q.[6] For all the remaining values in Q, we define δ_i by

$$k_i + a_i = A'(\gamma_i) + \delta_i.$$

Note that, by assumption, there exists an index $j \in \mathcal{H}$ and $j > d+2$ such that $\delta_j \neq 0$. To simplify the notation, let $\mathcal{H}_1$ be the first $d+1$ indexes of the honest servers, i.e. those used for defining polynomial $A'(x)$, and let $\mathcal{H}_2$ be the remaining honest servers.

On the other hand, for the corrupt servers, one can always fix a view that is consistent with $A'(x)$ and with the interaction with (honest) Bob, exactly as in the proof of the previous lemma. Thus we assume (for notational convenience) that a_i for $i \in \mathcal{C}$ is defined this way such that $A'(\gamma_i) = a_i$.

Now, from the communication in the computation phase, for each index i the outcome for Alice and Bob consists of two field elements $C_1(\gamma_i)$ and $C_2(\gamma_i)$ (of course, corrupt Alice does not necessarily store the $C_1(\gamma_i)$'s, all we mean to

[6] The choice to interpolate from the first $d+1$ values is completely arbitrary, the following argument will work no matter the choice of subset.

say is that they can be computed from the adversary's view). Now, from our definition of $A'(x)$ and δ_i and the protocol specification, it is easy to see that $C_1(\gamma_i) + C_2(\gamma_i) = A'(\gamma_i)B(\gamma_i)$ if $i \in \mathcal{C} \cup \mathcal{H}_1$, and otherwise $C_1(\gamma_i) + C_2(\gamma_i) = (A'(\gamma_i)+\delta_i)B(\gamma_i) = A'(\gamma_i)B(\gamma_i)+\delta_i B(\gamma_i)$ if $i \in \mathcal{H}_2$. For notational convenience, we will define $\delta_i = 0$ for $i \in \mathcal{C} \cup \mathcal{H}_1$, so we have

$$C_1(\gamma_i) + C_2(\gamma_i) = A'(\gamma_i)B(\gamma_i) + \delta_i B(\gamma_i) \text{ for } i \in [n]$$

We can now interpolate polynomials of degree at most $2d$ from the $C_1(\gamma_i)$'s, the $C_2(\gamma_i)$'s and the $C_1(\gamma_i) + C_2(\gamma_i) = A'(\gamma_i)B(\gamma_i) + \delta_i B(\gamma_i)$-values. Because interpolation is linear, this results in polynomials $C_1(x)$, $C_2(x)$ such that

$$C_1(x) + C_2(x) = A'(x)B(x) + \mathit{\Delta}(x), \text{ where } \mathit{\Delta}(\gamma_i) = B(\gamma_i)\delta_i \text{ for } i \in [n].$$

In the test phase, Bob sends a point t_B and Alice returns two field elements, that are "supposed to be" $A'(t_B)$ and $C_1(t_B)$. We can always write what she actually sends as $\alpha + A'(t_B)$ and $\beta + C_1(t_B)$, for some α, β that the adversary can compute. Bob will check the equation

$$(\beta + C_1(t_B)) + C_2(t_B) = (\alpha + A'(t_B))B(t_B)$$

which easily simplifies to $\beta + \mathit{\Delta}(t_B) = \alpha B(t_B)$.

So what we need to argue is that the adversary can guess α, β satisfying this equation with only negligible probability. This will turn out to be because he does not have sufficient information about the polynomial $B(x)$. Note that the adversary has seen t values of $B(x)$ from Bob's interaction with the corrupt servers ($B(t_A)$ has been committed to but is not revealed yet). Since the degree is $d = t + m$ and $m \geq 1$, the adversary is at least 2 points short of being able to determine $B(x)$. We can therefore assume that the values $B(t_B)$ and $B(\gamma_j)$ are independent and uniform in the view of the adversary, namely γ_j is the index of an honest server and t_B is chosen such that it is never the index of a corrupt server. To emphasize that the values are unknown, we will write $X = B(t_B)$ and $Y = B(\gamma_j)$.

We can imagine giving the adversary extra points so he knows exactly $d-1$ points on $B(x)$, this can only help him. Therefore, using the formulas of interpolation, the adversary can write any value of $B(x)$ as an affine linear combination of X and Y with known coefficients. In particular, this means there exist field elements $\omega_i, \eta_i, \sigma_i$ such that $\mathit{\Delta}(\gamma_i) = B(\gamma_i)\delta_i = \omega_i X + \eta_i Y + \sigma_i$. From the values $\omega_1, ..., \omega_n$ we can interpolate exactly one polynomial of degree at most $2d$, which we call $F(x)$. Likewise we interpolate $G(x)$ from the η_i's and $H(x)$ from the σ_i's. This immediately implies that

$$\mathit{\Delta}(x) = F(x)X + G(x)Y + H(x)$$

The meaning of this equation is that the polynomials $F(x), G(x)$ and $H(x)$ are fixed in the sense that they do not depend on the choice of $B(x)$ (and hence of X, Y). So in the adversary's view, the polynomial $\mathit{\Delta}(x)$ depends linearly on the two (random) values X, Y as described by the equation.

Note that for $i = j$, we have $\delta_j B(\gamma_j) = \delta_j Y = \omega_j X + \eta_j Y + \sigma_j$ which can only be true for random X and Y if $\omega_j = \sigma_j = 0$ and $\eta_j = \delta_j \neq 0$. So this implies that $G(x)$ is not the 0-polynomial. Hence the above equation that the adversary must try to satisfy becomes:

$$\beta + F(t_B)X + G(t_B)Y + H(t_B) = \alpha X$$

Which can be rewritten as $(F(t_B) - \alpha)X + G(t_B)Y + \beta + H(t_B) = 0$.

Note that we can think of the experiment done as follows: first we choose t_B at random from a set of size $|\mathbb{F}| - (m + n)$, then the adversary chooses α, β and then we choose X, Y independently and uniformly at random in $\mathbb{F}$. It is then clear that if $(F(t_B) - \alpha) \neq 0$ then the left-hand side is uniformly random and so is 0 with probability $1/|\mathbb{F}|$. On the other hand, if $(F(t_B) - \alpha) = 0$, we can use the fact that $G(x)$ is non-zero and has degree at most $2d$ to conclude that $G(t_B) = 0$ with probability at most $2d/(|\mathbb{F}| - (m + n))$. But if $G(t_B) \neq 0$, then again the left-hand side is uniformly random and is 0 with probability $1/|\mathbb{F}|$. We conclude that the equation is satisfied with probability at most

$$\frac{2d}{|\mathbb{F}| - (m+n)} + \frac{1}{|\mathbb{F}|} \leq \frac{n}{|\mathbb{F}| - (m+n)}$$

□

Having proved the lemmas, we present a simulator $\mathcal{S}_A$ (see Simulator 1) which provides statistically indistinguishable simulation of the protocol π_{ROLE} against a malicious adversary that corrupts Alice and t servers. The simulator basically runs its own instance of the protocol with corrupt Alice and servers, playing honestly for Bob and the honest servers. There is, however, an important difference: During the check phase, the simulator aborts under a different condition than in the real protocol: While an honest Bob in the real protocol aborts only if the values sent by Alice do not satisfy the right relation with the polynomials held by Bob, the simulator will also abort if Alice does not behave consistently. Now, there two cases to consider:

1. Alice behaves consistently: in this case, the simulator follows the protocol until the end, so it is clear that simulation of the adversary's view of the protocol is perfect. Furthermore, it follows by Lemma 2 that the simulator extracts the only possible candidate for Alice's output shares, given the interaction with honest players, so what it sends to the functionality is correct. Hence the only difference between the real and the ideal process is that in the real process, Bob's output is extracted from his view of the protocol, whereas in the ideal process it is chosen by the functionality (but consistently with Alice's shares). This makes no difference: Alice has seen $t + 1$ points on the polynomial $B(x)$ and since the degree is $d = t + m$ this leaves m degrees of freedom which means that the values $B(\sigma_1), ..., B(\sigma_m)$ are random and independent of the adversary/environment's view of the protocol. So in this case, the real and ideal process are perfectly indistinguishable.

2. Alice does not behave consistently: in this case the ideal process always aborts, but by Lemma 3 the real process does the same, except with negligible probability. Thus in this case, the processes are statistically indistinguishable. □

Simulator 1: Simulator $\mathcal{S}_A$ against corrupt Alice

```
The simulator starts by initializing copies of the code for the honest servers, for
honest Bob, and for $\mathcal{F}_{\text{COMMIT}}$. Then the simulation proceeds as follows:

Stateless oblivious RPC and Computation phase:
 1. Let the simulator's copies of the honest servers and Bob interact with corrupt
    Alice and the corrupt servers (which are controlled by the environment).
 2. When the computation phase is done, check whether Alice has acted consis-
    tently (see Definition 3).
 3. If Alice has not acted consistently, set a flag will-abort = true. Else (Alice
    has acted consistently), do as follows:
    (a) Compute polynomials $A^*(x), C_1^*(x)$ as guaranteed by Lemma 2.
    (b) Compute $\hat{\boldsymbol{u}} = (A^*(\sigma_1), ..., A^*(\sigma_m))$. $\hat{\boldsymbol{w}}_1 = (C_1^*(\sigma_1), ..., C_1^*(\sigma_m))$.
        Send $(\texttt{corrupt}, \texttt{A}, (\hat{\boldsymbol{u}}, \hat{\boldsymbol{w}}_1))$ to the ideal functionality $\mathcal{F}^m_{\text{ROLE}}$;
Check phase:
 1. Let the simulator's copies of Bob and $\mathcal{F}_{\text{COMMIT}}$ do the check phase with corrupt
    Alice. If the test done by Bob fails, set the flag will-abort = true.
Output phase:
 1. If will-abort = true, abort the protocol.
    Else, send $(\texttt{deliver}, \texttt{A})$ and $(\texttt{deliver}, \texttt{B})$ to the ideal functionality $\mathcal{F}^m_{\text{ROLE}}$.
```

4 Implementation

We implement and measure timings for the commodity-based OLE protocol.[7] The two clients Alice and Bob are set up on a basic LAN and will connect to some number of servers around the world. Since the experiments are identical up to the output for different tradeoff choices of t and m, we implement the setting of generating one OLE $m = 1$ with maximum adversarial threshold $t \leq \frac{n-3}{2}$.

Instantiations. We use a basic OpenSSL (version 1.1.0) setup to implement a public key infrastructure for the clients to authenticate servers. Our setting consists of a single root certificate authority, trusted by each client, and who have signed certificates to each server. This setup is easily used in the real world, as two clients can simply agree on some domain name and rely on root certificates already included in the system to do hostname validation.

All finite fields are implemented using GNU Multiple Precision Arithmetic Library and when testing a b-bit prime field, we refer to $\mathbb{Z}_p$ for the largest $p < 2^b$. We instantiate the hash function $\mathcal{H}$ as SHA256. For sampling random numbers,

[7] The sources used for the benchmark implementation are available at `http://orlandi.dk/commodity`.

we construct a PRG by using the AES instruction set in counter mode. This PRG takes a seed s of arbitrary length and set the AES key to be the 128 first bits of $\mathcal{H}(s)$. To generate a random field element from $\mathbb{Z}_p$, we sample $b = \lceil \log_2 p \rceil$ random bits from the PRG repeatedly until it represents a valid element.

We fix the polynomial evaluation points $\gamma_1, \ldots, \gamma_n$ to be $1, \ldots, n$, and set the extraction point $\sigma_1 = 0$. We use the following preprocessed variant of Lagrange interpolation to fast[8] evaluate $f(x)$ where f is a degree d polynomial represented by $d+1$ points $y_1, \ldots, y_n$ and $y_i = f(i)$. First preprocess $\delta_{ij} = (i-j)^{-1}$ and $\lambda_{ij} = j \cdot \delta_{ij}$ for $i, j \in [d+1]$, and then compute the point $f(x)$ as:

$$f(x) = \sum_{i=1}^{d+1} y_i \prod_{\substack{j=1 \\ j \neq i}}^{d+1} x\delta_{ij} - \lambda_{ij}$$

Set-up and results. The two clients are tested on a basic LAN setup consisting of two identical machines each with a i7-3770K CPU running at 3.5GHz, 32GB of RAM and connected via 1GbE with a 0.15ms delay. The servers are set up on Amazon Cloud using M4.LARGE instances with 2 vCPUs and are spread across five continents namely North America (N. Virginia), South America (São Paulo), Europe (Ireland), Asia (Mumbai) and Australia (Sydney). The Internet connection for all servers and clients was measured to vary between 200-500 Mbps up and down at the time of testing.

We test the protocol with different field sizes ranging from 32 to 2048 bit and tolerating up to $1, 5$ or 10 malicious servers colluding with one malicious client. This implies the number of servers used in each setting being $5, 13$ and 23 respectively. Both clients run a producer-consumer program where the producer is connected to all servers and produce batches of shares from each server to be used for the protocol. The consumer is connected to the other client and consumes a batch by running the protocol in parallel for each OLE to be corrected. All measurements are done as an average over 30 seconds.

First we measure sequential timings for protocol, namely how long time a single consumer (thread) takes to compute a corrected ROLE given the raw material. We test this in two versions, one where the clients are only interested in a single ROLE and another where they want a batch of 1000. The first version may be interesting in a real-world application where clients wants a single OLE for say a commitment, and serves well as a baseline for OLE protocol comparison. The second version on the contrary shows what to expect, if our protocol is to be used in a subsequent protocol requiring 1000 OLEs. Here, one would expect a batch of 1000 to take 1000 times as long – however the numbers show that this vary between roughly 200 to 900 depending on the field size and number of servers, which is partly due to less network delay. Finally, amortized timings for the protocol are measured. These timings show how many ROLEs we can generate per second. We simply let the machines generate as many ROLEs as

[8] Fast in practice for low-degree polynomials, but theoretically inferior to the Fast Fourier Transform.

possible by turning up the number of consumers. Note that our tests was done on a university network and on shared cloud nodes, which meant inconsistency in available resources. This, together with different parameter choices to maximize parallization (number of threads and batch size), means that we can expect to see jumps in the amortized table, for example for 23 servers and $b = 256$ and $b = 512$.

	$n = 5$	$n = 13$	$n = 23$
	$t = 1$	$t = 5$	$t = 10$
$b = 32$	0.301ms	0.460ms	0.930ms
$b = 64$	0.317ms	0.465ms	1.162ms
$b = 128$	0.333ms	0.803ms	1.294ms
$b = 256$	0.980ms	1.388ms	2.311ms
$b = 512$	1.372ms	1.891ms	3.301ms
$b = 1024$	1.491ms	2.625ms	5.228ms
$b = 2048$	1.856ms	4.252ms	9.311ms

Table 1. Sequential timings for one OLE

	$n = 5$	$n = 13$	$n = 23$
	$t = 1$	$t = 5$	$t = 10$
$b = 32$	13.450ms	86.103ms	370.984ms
$b = 64$	13.356ms	136.978ms	405.691ms
$b = 128$	15.334ms	171.623ms	468.540ms
$b = 256$	24.716ms	317.403ms	829.741ms
$b = 512$	51.997ms	410.123ms	1792.491ms
$b = 1024$	151.008ms	858.411ms	2820.248ms
$b = 2048$	371.269ms	2188.804ms	7970.499ms

Table 2. Sequential timings for batch of 1000 OLEs

	$n = 5$	$n = 13$	$n = 23$
	$t = 1$	$t = 5$	$t = 10$
$b = 32$	3.570μs	23.657μs	89.055μs
$b = 64$	5.801μs	23.362μs	105.628μs
$b = 128$	16.867μs	28.201μs	106.868μs
$b = 256$	33.101μs	56.197μs	191.985μs
$b = 512$	69.297μs	115.180μs	2160.664μs
$b = 1024$	118.353μs	230.487μs	4938.348μs
$b = 2048$	249.018μs	516.934μs	7709.943μs

Table 3. Amortized timing for generating one OLE

Using the tradeoff, one can increase m and decrease t to get a protocol with same sequential running time, but with higher throughput i.e. lower amortized timings. For example, the second column for the amortized timings represent $n = 13, t = 5$ and $m = 1$ – but we can get a five time increase in throughput by running the protocol with $n = 13, t = 1$ and $m = 5$. Likewise, for the case of $m = 2$, one can decrease t by one to obtain a column with half the amortized timings.

In the case of 128-bit fields, existing protocols providing computational security like the “Overdrive” protocol [KPR18] achieve a secure 128-bit multiplication in roughly 0.03 msec. To compare our protocol roughly, one can set $n = 13$ and assume a 14 times overhead by using the optimized[9] TinyOLE variant, we can compute rough a secure multiplication between 0.4 and 0.08 msec with a choice of $(t = 5, m = 1)$ and $(t = 1, m = 5)$ respectively. We stress the difference between the models used by us and “Overdrive”: ours provide unconditional security (assuming $\leq t$ malicious servers colluding) rather than computational security - and we believe these numbers show that our protocol is indeed applicable in a practical setting.

5 Commodity-based Multiplication Triples

In this section we investigate the construction of authenticated multiplication triples in the commodity model. Multiplication triples[10] as introduced by Beaver [Bea91] are complete for multiparty computation, and if all values are authenticated under a global key, these triples allow for a fast dishonest-majority multiparty computation as presented in [BDOZ11,DPSZ12].

We explore whether commodity-based protocols exists for producing authenticated multiplication triples, that are more efficient than using known reductions from OLE or OT, and show this in the affirmative. However, this efficiency comes with a cost in security, as we can no longer tolerate a malicious client being in control of a server – more formally, the adversary will be limited to corrupting one client *or* up to t servers.

Our protocol allows clients to securely combine multiplication triples from n servers into a single sound one. The high-level idea of the protocol closely follows the structure from Protocol 1 with some variation. The clients first obtain random triples authenticated under different global keys from the servers, which makes up the stateless oblivious RPC. The main issue compared to our OLE protocol is that the MACs associated to the authenticated triples are computed using different keys (delivered from each server), and therefore the triples cannot be combined. To deal with this, we note that the MACs are *key-homomorphic* and therefore the clients can adjust the MAC keys with one round of interaction. In more detail, all MACs are of the form $t = \alpha x + \beta$ where one party knows the value x and the MAC t, while the other party knows the key (α, β). The clients can now turn this into a MAC using the key (α', β) using the following protocol: the key owner sends the shift value $\Delta = \alpha' - \alpha$, and the other party adjusts the MAC to be $t' = t + \Delta x = \alpha' x + \beta$.

Once the MACs are adjusted to use the same keys, the clients can combine the triples they receive from the servers to ensure *privacy* and *correctness*. This

[9] Which requires 14 OLEs to produce a secure multiplication, by personal communication

[10] The primitive for (non-authenticated) multiplication triples provides Alice with a_A, b_A, c_A and Bob a_B, b_B, c_B such that $(a_A + a_B)(b_A + b_B) = c_A + c_B$). We also say the parties jointly hold $[a][b] = [c]$ where $a = a_A + a_B, b = b_A + b_B$ and $c = c_A + c_B$.

reduction closely follows our OLE protocol, but we error-correct authenticated secret shares $[\![x]\!]$ (defined below) instead of the shares from the OLE primitive. That is, first the triples are combined to define two authenticated secret shared degree d polynomials $[\![U(X)]\!]$ and $[\![V(X)]\!]$, and points on an authenticated secret shared degree $2d$ polynomial $[\![W(X)]\!]$. The other half of the triples are then used to evaluate $[\![U(X)]\!] \cdot [\![V(X)]\!]$ on agreed upon points until enough points are obtained to fix $[\![W(X)]\!]$. After the construction of these polynomials, the clients perform a check that with high probability reveals whether the polynomials are valid i.e. $[\![U(\gamma)]\!] \cdot [\![V(\gamma)]\!] = [\![W(\gamma)]\!]$ hold for all $\gamma \in \mathbb{F}$. The final random triple that the parties return will be defined by a fixed point $\gamma_0 \in \mathbb{F}$ on these polynomials:

$$\Big([\![u]\!], [\![v]\!], [\![w]\!]\Big) = \Big([\![U(\gamma_0)]\!], [\![V(\gamma_0)]\!], [\![W(\gamma_0)]\!]\Big)$$

5.1 Authenticated Secret Sharing

We will use the notation for authenticated secret shared variables over $\mathbb{F}$ as defined in [BDOZ11,DPSZ12], and make use of the same linear MAC scheme and operations over such secret shares. We extend the notation such that each party knows a set of global keys and a set of authenticated secret shares for each of these global keys.

Definition 4. *Let Alice hold random global keys* $\{\alpha_A^1, \dots, \alpha_A^n\} \xleftarrow{\$} \mathbb{F}^n$*, and Bob likewise* $\{\alpha_B^1, \dots, \alpha_B^n\} \xleftarrow{\$} \mathbb{F}^n$*. Define* $[\![x]\!]^i$ *as an authenticated secret sharing of* $x \in \mathbb{F}$ *under global keys* α_A^i *and* α_B^i*, where Alice hold* $[\![x]\!]_A^i = (x_A, t_A^x, \beta_A^x)$ *and Bob hold* $[\![x]\!]_B^i = (x_B, t_B^x, \beta_B^x)$ *such that:*

- $x = x_A + x_B$
- $\beta_A^x, \beta_B^x \xleftarrow{\$} \mathbb{F}$
- $t_A^x = \alpha_B^i x_A + \beta_B^x$ *and* $t_B^x = \alpha_A^i x_B + \beta_A^x$

Define a random sharing as $[\![r]\!]$ for uniformly random $r \xleftarrow{\$} \mathbb{F}$, and define an authenticated multiplication triple for key $i \in [n]$ as three secret shared variables of random $a, b, c \xleftarrow{\$} \mathbb{F}$ under the same global key where it holds that $ab = c$. That is, the parties hold random $[\![a]\!]^i, [\![b]\!]^i, [\![c]\!]^i$ for which $[\![ab]\!]^i = [\![c]\!]^i$. In Figure 4 is presented the ideal functionality for authenticated multiplication triples.

5.2 Computation on Shared Secrets

Given authenticated secret sharings $[\![x]\!]^i$ and $[\![y]\!]^i$ for $i \in [n]$, the parties can compute different operations on the shares. We will make use of these operations in our protocol description.

ADDITION: Let $[\![z]\!]^i = [\![x+y]\!]^i = [\![x]\!]^i + [\![y]\!]^i$ denote local addition, where each client adds the triples component-wise, that is Alice sets $z_A = x_A + y_A$, $t_A^z = t_A^x + t_A^y$, and $\beta_A^z = \beta_A^x + \beta_A^y$ (likewise for Bob).

Functionality $\mathcal{F}_{\text{TRIPLE}}$

1. Upon receiving message $(\texttt{corrupt}, \texttt{A}, (u_A, v_A, w_A, \alpha_A))$ on the influence port: if no values has been stored then
   - draw uniformly random $u_B, v_B \in \mathbb{F}$;
   - compute $w_B := (u_A + u_B)(v_A + v_B) - w_A$;
   - draw uniformly random $\alpha_B \in_R \mathbb{F}$ and for all $x \in \{u, v, w\}$ draw uniformly random $\beta_A^x, \beta_B^x \in_R \mathbb{F}$;
   - compute the MACs;
   - store $(\llbracket u \rrbracket, \llbracket v \rrbracket, \llbracket w \rrbracket, \alpha_A, \alpha_B)$.

   (Likewise upon receiving message $(\texttt{corrupt}, \texttt{B}, (u_B, v_B, w_B, \alpha_B))$.)
2. Upon receiving message $(\texttt{deliver}, \texttt{A})$ on the influence port: if no values has been stored then
   - draw uniformly random $u_A, u_B, v_A, v_B, w_A \in \mathbb{F}$
   - compute $w_B := (u_A + u_B)(v_A + v_B) - w_A$
   - draw uniformly random $\alpha_A, \alpha_B, \beta_A^x, \beta_B^x \in_R \mathbb{F}$ for all $x \in \{u, v, w\}$;
   - compute the MACs;
   - store $(\llbracket u \rrbracket, \llbracket v \rrbracket, \llbracket w \rrbracket, \alpha_A, \alpha_B)$.

   Send $(\texttt{output}, (\llbracket u \rrbracket_A, \llbracket v \rrbracket_A, \llbracket w \rrbracket_A, \alpha_A))$ to Alice.
   (Likewise upon receiving message $(\texttt{deliver}, \texttt{B})$.)

Fig. 4. Ideal Functionality $\mathcal{F}_{\text{TRIPLE}}$ for Authenticated Multiplication Triples.

Opening: Let $(x, \perp) = \mathsf{Open}(A, \llbracket x \rrbracket^i)$ denote the opening of a secret shared value to Alice, where Bob sends $\llbracket x \rrbracket_B^i$ to Alice, who reconstruct the message $x = x_A + x_B$ and check the MAC. Likewise, we define $(\perp, x) = \mathsf{Open}(B, \llbracket x \rrbracket^i)$ to denote the opening of a secret shared value to Bob. We write $x = \mathsf{OpenAll}(\llbracket x \rrbracket^i)$ as a shorthand for $(x, \perp) = \mathsf{Open}(A, \llbracket x \rrbracket^i)$ and $(\perp, x) = \mathsf{Open}(B, \llbracket x \rrbracket^i)$.

Multiplication: Given a multiplication triple, i.e. random $\llbracket a \rrbracket^i, \llbracket b \rrbracket^i$ and $\llbracket c \rrbracket^i$ such that $ab = c$, then the parties can compute

$$\llbracket z \rrbracket^i = \llbracket xy \rrbracket^i = \mathsf{Mul}(\llbracket x \rrbracket^i, \llbracket y \rrbracket^i, \llbracket a \rrbracket^i, \llbracket b \rrbracket^i, \llbracket c \rrbracket^i)$$

where Mul is the following subprotocol:

- Open two values $e = \mathsf{OpenAll}(\llbracket a \rrbracket^i + \llbracket x \rrbracket^i)$ and $d = \mathsf{OpenAll}(\llbracket b \rrbracket^i + \llbracket y \rrbracket^i)$
- Locally compute the product $\llbracket z \rrbracket^i = \llbracket c \rrbracket^i + e\llbracket y \rrbracket^i + d\llbracket x \rrbracket^i - ed$.

Key Shift: Note that for the above operations to work, we need that the authenticated sharings are defined under the same global key. Thus, we define a key shift operation that allows the parties to change the global key of their sharings in order to get one common global key. On input keys $(\alpha_A, \alpha_A^i), (\alpha_B, \alpha_B^i)$ and set of authenticated secret sharing $\{\llbracket x_\ell \rrbracket^i\}_\ell$ under global keys α_A^i, α_B^i, we define the key shift operation

$$\{\llbracket x_\ell \rrbracket\}_\ell \leftarrow \pi_{\text{KeyShift}}\left(\{\llbracket x_\ell \rrbracket^i\}_\ell, (\alpha_A, \alpha_A^i), (\alpha_B, \alpha_B^i)\right)$$

as the following subprotocol between Alice and Bob:

- Alice knows $\{[\![x_\ell]\!]_A^i\}_\ell, \alpha_A, \alpha_A^i$
- Alice sends $\Delta_A = \alpha_A - \alpha_A^i$ to Bob
- For all ℓ Bob outputs $[\![x_\ell]\!]_B = ((x_\ell)_B, t_B^{(x_\ell)} + (x_\ell)_B \cdot \Delta_A, \beta_B^{(x_\ell)})$

(Likewise, Bob sends Δ_B to Alice).

5.3 Protocol for Multiplication Triples

The protocol for commodity-based authenticated multiplication triples is given as:

Protocol π_{TRIPLE}

Public: Let $n = 2d+1$. Let $\mathcal{T} = \{\gamma_0, \dots, \gamma_n\}$ be a set of publicly known distinct points in $\mathbb{F}$. Let $\{\mathtt{S}_1, \dots, \mathtt{S}_n\}$ be the set of servers and {Alice, Bob} the clients.
Output: A secret shared triple $([\![u]\!], [\![v]\!], [\![w]\!])$ such that $u \cdot v = w$.
Protocol:
Stateless oblivious RPC:
1. The clients sends a request to each server $\mathtt{S}_i$ for $i \in [n]$.
2. Server $\mathtt{S}_i$ for $i \in [n]$:
   - choose random global keys α_A^i and α_B^i.
   - creates a random authenticated secret shared triple $[\![a_i]\!]^i, [\![b_i]\!]^i, [\![c_i]\!]^i$ such that $a_i \cdot b_i = c_i$ and $a_i, b_i, c_i \in_R \mathbb{F}$.
   - send $[\![a_i]\!]_A^i, [\![b_i]\!]_A^i, [\![c_i]\!]_A^i$ and global key α_A^i to Alice.
   - send $[\![a_i]\!]_B^i, [\![b_i]\!]_B^i, [\![c_i]\!]_B^i$ and global key α_B^i to Bob.

Computation phase:

3. The clients each sample new random global keys: α_A and α_B
4. For all $i \in [n]$:
$$([\![a_i]\!], [\![b_i]\!], [\![c_i]\!]) \leftarrow \pi_{\text{KeyShift}}\Big(([\![a_i]\!]^i, [\![b_i]\!]^i, [\![c_i]\!]^i), (\alpha_A, \alpha_A^i), (\alpha_B, \alpha_B^i)\Big)$$
   i.e. the clients jointly perform the key shift protocol for all authenticated secret sharings they received from the servers.
5. The clients jointly interpolate three secret shared polynomials:
   - $[\![U(X)]\!]$ of degree d by setting $[\![U(\gamma_i)]\!] = [\![a_i]\!]$ for all $i \in [d+1]$.
   - $[\![V(X)]\!]$ of degree d by setting $[\![V(\gamma_i)]\!] = [\![b_i]\!]$ for all $i \in [d+1]$.
   - $[\![W(X)]\!]$ of degree $2d$ by setting
$$[\![W(\gamma_i)]\!] = [\![c_i]\!] \quad \text{for } i = 1, \dots, d+1$$
$$[\![W(\gamma_i)]\!] = \mathsf{Mul}([\![U(\gamma_i)]\!], [\![V(\gamma_i)]\!], [\![a_i]\!], [\![b_i]\!], [\![c_i]\!]) \quad \text{for } i = d+2, \dots, n$$

Check phase:

6. Alice: draw $t_A \in_R \mathbb{F} \backslash \mathcal{T}$ uniformly at random and send t_A to Bob.
7. Bob:
   - draw $t_B \in_R \mathbb{F} \backslash \mathcal{T}$ uniformly at random and send t_B to Alice.

– perform openings to Alice:

$$(U(t_A), \perp) = \mathsf{Open}(A, [\![U(t_A)]\!])$$
$$(V(t_A), \perp) = \mathsf{Open}(A, [\![V(t_A)]\!])$$
$$(W(t_A), \perp) = \mathsf{Open}(A, [\![W(t_A)]\!])$$

8. Alice:
   – check if $U(t_A) \cdot V(t_A) = W(t_A)$ and abort if not.
   – perform openings to Bob:

$$(\perp, U(t_B)) = \mathsf{Open}(B, [\![U(t_B)]\!])$$
$$(\perp, V(t_B)) = \mathsf{Open}(B, [\![V(t_B)]\!])$$
$$(\perp, W(t_B)) = \mathsf{Open}(B, [\![W(t_B)]\!])$$

9. Bob: check if $U(t_B) \cdot V(t_B) = W(t_B)$ and abort if not.

Output phase:

10. The clients outputs $([\![u]\!], [\![v]\!], [\![w]\!]) = ([\![U(\gamma_0)]\!], [\![V(\gamma_0)]\!], [\![W(\gamma_0)]\!])$.

Protocol 2: Protocol for commodity-based authenticated multiplication triples

Theorem 2. *The protocol π_{TRIPLE} is a UC-secure implementation of $\mathcal{F}_{\text{TRIPLE}}$ in the commodity-model with statistical security against any active, static adversaries, corrupting a single client (with $n \geq 3$)* or $t \leq \frac{n-1}{2}$ *servers.*

Proof. We first consider an adversary that corrupts one of the two clients, say Alice without loss of generality (similarly to the OLE protocol, this protocol is completely symmetric except for the check phase, in which Alice learns Bob's check point before sending its point. Thus, the harder case is to prove security when Alice is corrupt, and the case of Bob follows in a straightforward way). This case is relatively easy and reminiscent of the proof of most protocols in the preprocessing model such as the [BDOZ11] protocol: note that, in this case, the simulator controls *all* the servers. This implies that at all stages of the simulation the simulator knows exactly which values the corrupt client is supposed to hold (note that the protocol is almost deterministic given the preprocessing material, with the only exception of the choice of the new global MAC key α_A and the check point t_A). Therefore, the simulator runs the protocol as the honest servers/ client would do with a single but important difference: instead of checking the values received from the corrupt client using the MAC keys held by the honest client, the simulator will simply abort if any of the received values (or MACs) is not the value which the client was supposed to send. Due to the (statistical) security of the MACs, we easily argue for indistinguishability.

In more details, the simulator proceeds in the following way: the simulator samples random triples and keys for the corrupt client (as the honest servers would do) in the RPC phase. Note that the simulator does not need to sample the shares for the honest client, and that these values will stay undefined during the simulation, thus we will be able to argue for privacy. In the computation phase, the simulator receives the values Δ_A^i as part of the key-shift subprotocol.

Since the simulator knows the values α_A^i, the simulator can compute the set $\{\Delta_A^i + \alpha_A^i\}$ for all i's. In case the set is a singleton then the simulator extracts the key α'_A chosen by the corrupt client, otherwise the simulator notices that the client is deviating from the protocol, but will not abort. Instead, in the case where the set is not a singleton, the simulator defines $\alpha'_A = \Delta_A^1 + \alpha_A^1$ (e.g., arbitrarily as the key defined by the first tuple), and then define a vector of errors $\varepsilon_i = \alpha'_A - (\Delta_A^i + \alpha_A^i)$ for all $i \in [n]$.

When simulating the multiplication subprotocols necessary to generate the polynomial W, the simulator will send random shares on behalf of the honest client and will abort if it receives any invalid authentication tag or a wrong value.

In the check phase the simulator will receive t_A and "fake" the open subprotocols to output three random values $U(t_A), V(t_A)$ and $W(t_A) = U(t_A)V(t_A)$. Finally the simulator receives the values for the open subprotocols from Alice and aborts if any of the values received are not the ones predicted by the simulator.

To argue for indistinguishability we note that the client only sees random values, exactly as in the execution of the real protocol and that, if there are more honest servers than point on the polynomials disclosed to the adversary, the extracted point in the real protocol and in the ideal world will have identical distribution. Note that during the protocol even an honest client will learn one point on the polynomial, "leaked" as a result of the check phase. In addition we note that a corrupt client can learn an extra point on the polynomial by cheating during the key-shift phase, in such a way that the vector $(\varepsilon_1, \ldots, \varepsilon_n) \neq (0, \ldots, 0)$. When this is the case the MAC that the honest client sends to the corrupted client is not only a function of the final values $U(t_A), V(t_A), W(t_A)$, but of the initial values a_i, b_i, c_i as well. Consider for instance the value $z = U(t_A)$: the value t_A defines coefficients $\delta_1, \ldots, \delta_{d+1}$ such that $z = \sum \delta_i a_i$. The MAC on this value during an correct execution of the protocol will be the value $t_1^z = \alpha_1 U(t_A) + \beta_1^z$. However if Alice is corrupt and sends inconsistent values of Δ_1^i (during the key-shift protocol for the different triples (a_i, b_i, c_i)), the difference between t_1^z (the MAC that would be sent in an honest execution) and the MAC resulting from a malicious execution is $\tau = \sum \delta_i \varepsilon_i a_i$, which leaks one extra constraint on the coefficient of the polynomial. It is easy to see that leaking τ is not worse than leaking to the adversary one of the a_i's. Therefore, as the corrupt client learns at most two points (one via the check, and potentially one if the corrupt client cheats in the key-shift phase), the security of the protocol requires the number of servers to be at least $n \geq 3$.

We now argue for security when a minority $t \leq (n-1)/2$ of the servers are actively corrupted. We need to argue that an environment controlling this number of servers will not be able to distinguish between the output of the ideal functionality in the simulated execution and the output in a real execution. In this case the simulator receives the data from the corrupt servers on behalf of both the honest clients. In a nutshell, we argue for privacy of the output triple since even if the adversary knows $t \leq d$ tuples (those generated by the corrupted servers), this still gives no information about the degree $\geq d$ polynomials U, V, W. To argue for correctness, we let the simulator abort immediately if the commodities received from the corrupt servers are incorrect. To conclude the proof it is therefore sufficient to argue that the two honest clients will also abort in case the commodities are incorrect except with negligible probability: it is trivial to see that if any of the MACs or relative key material sent by the corrupt servers is incorrect, then the protocol will abort with probability $1 - 1/|\mathbb{F}|$. Assume now that for some $c_i = a_i b_i + \varepsilon_i$ for at least one i i.e., some of the tuples sent by the adversary are not correct multiplicative triples. This implies that the resulting polynomial $W(X) \neq U(X)V(X)$ and therefore by the fundamental theorem of algebra the probability that the two polynomials will be equal on the check points t_A, t_B is negligible in the field size. $\square$

6 Allowing Servers to be Memoryless

We now look at practical aspect of the commodity model. As mentioned in the introduction, it will clearly be an advantage if the servers in our model do not need memory in the sense that they do not have to remember who they talked to or what was sent. This would mean that a server will not have to administrate identities and log-in credentials, and could simply generate the required data on the fly, say, against payment in some cryptocurrency. An obvious problem with this goal is that the data sent to two clients Alice and Bob must be correlated and a memoryless server cannot make this happen on its own. We now informally sketch a solution to this: We will assume that clients can communicate with servers such that clients can authenticate the identity of servers, but not necessarily the other way around (in practice, one may think of 1-way TLS here.)

We also assume that Alice and Bob interact before talking to the servers – indeed this is necessary to create any correlation if the servers have no memory. We assume a 2-way authenticated channel for this, indeed this seems necessary if there are several clients, otherwise an honest Alice could not know with whom she is doing secure computation. Alice and Bob would then agree on a common nonce n_{AB}, as well as a parameter par specifying what they will request from the server, as well as the identity id of the server. For the case of our protocol, we would have $par = (\mathbb{F}, s, id)$ where $\mathbb{F}$ is the field to use for the OLEs, s is the number of OLEs to request and id is the server identifier.

In a naive solution, Alice would send (“A”, par, n_{AB}) to the server who will use this and a private PRF key K to generate Alice’s data, and something similar is done for Bob. However, this is clearly not secure: if Alice is corrupt, she can send both (“A”, par, n_{AB}) and (“B”, par, n_{AB}) to each honest server, get Bob’s data and break the protocol.

We solve this by generating the nonce such that Alice and Bob each know different secrets relating to the nonce and hence cannot impersonate the other party towards the server. In the simplest case where a nonce is used for only one server, a straightforward way to do this is to make use of a one-way function $f : \{0,1\}^k \mapsto \{0,1\}^k$ where k is a security parameter. Then Alice chooses $x_A \in \{0,1\}^k$ at random, similarly Bob chooses x_B and we let $n_{A,B} = f(x_A)||f(x_B)$ where $||$ denotes concatenation.

Now, party $P \in \{A, B\}$ would send ("P", par, x_P, n_{AB}). The server checks that x_P is correct with respect to n_{AB} and only then will it send data to P. In this case we can instantiate f efficiently using a hash function, for instance.

For the security of this solution, note that we just need to make sure that the data sent from an honest server to an honest client is secure, since all other data is known and can be modified by the adversary anyway. So assume Alice is honest and agrees on nonce n_{AB} and par with corrupt Bob, and let d_A be the data that honest server S will returns to Alice. Now, if Bob sends any request to S that contains something different from par, n_{AB} then S will return something that is (pseudo)uncorrelated to d_A. If the request does contain par, n_{AB}, then S may return either nothing or the data Bob is allowed to get, which is fine. It will only return d_A if the request contains x_A, and this happens only with negligible probability since f is one-way.

It is also possible to use one nonce for all servers. In that case we cannot let Alice simply reveal a preimage to the server. If the server is corrupt it can send x_A to Bob who can then do the same attack as before on honest servers. Instead we can let Alice generate a public key vk_A for a secure signature scheme, while Bob generates vk_B. Now, the request sent by Alice will be of form ("A", par, σ_A, n_{AB}), where σ_A is a signature on n_{AB} and par, and where $par = (\mathbb{F}, s, id, vk_A, vk_B)$ The server only sends back data if the signature verifies under the public key found in par, and if its own name occurs in par. Note that this last checks prevents a corrupt server from replaying a request to an honest server, and hence security can be argued in a similar way as before.

We remark that a practical implementation of any information theoretically secure MPC needs to implement the secure channels using encryption and (usually) a PKI, which is only computationally secure. We are in a similar situation, only we consider also the authentication aspect: if we assumed ideal authentic channels, the servers could generate data based on the ID's of the parties they know they are talking to. If we do not assume this, we have to use computational assumptions.

References

ADI$^+$17. Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 223–254. Springer, 2017.

BCD$^+$09. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, pages 325–343, 2009.

BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *Proceedings of the 2018 ACM SIGSAC Conference on*

Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018, pages 896–912, 2018.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 169–188, 2011.

Bea91. Donald Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *Proceedings of Crypto*, pages 420–432, Springer Verlag 1991.

Bea97. Donald Beaver. Commodity-based cryptography (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 446–455, 1997.

BLW08. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, pages 192–206, 2008.

Can01. Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*, pages 136–145, 2001.

DGN$^+$17. Nico Döttling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto Trifiletti. Tinyole: Efficient actively secure two-party computation from oblivious linear function evaluation. In *CCS*, pages 2263–2276. ACM, 2017.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

HIKN08. Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. Ot-combiners via secure computation. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, pages 393–411, 2008.

HKN$^+$05. Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 96–113, 2005.

IKM$^+$13. Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 600–620, 2013.

JNO14. Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW '14, Scottsdale, Arizona, USA, November 7, 2014*, pages 81–92, 2014.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM Conference on Computer and Communications Security*, pages 830–842. ACM, 2016.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT (3)*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189. Springer, 2018.

NP99. Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 245–254, 1999.

Rab05. Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

TND$^+$15. Rafael Tonicelli, Anderson C. A. Nascimento, Rafael Dowsley, Jörn Müller-Quade, Hideki Imai, Goichiro Hanaoka, and Akira Otsuka. Information-theoretically secure oblivious polynomial evaluation in the commodity-based model. *Int. J. Inf. Sec.*, 14(1):73–84, 2015.

A Our Constructions in the Big Picture

In this section we present a graphical overview of how our constructions fits in the bigger picture of performing secure 2-party computation.

Construction 1: Commodity-based OLEs. We construct commodity-based OLEs in the commitment-hybrid model with active, statistical security and in the standard model with active, information theoretic security (with slightly worse parameter). In this construction we allow a corrupt client to collude with a minority of the servers. Using the result from [DGN+17], we can use the commodity-based OLEs to construct authenticated multiplication triples, which are complete for multiparty computation.

Construction 2: Commodity-based Multiplication Triples. We construct commodity-based authenticated multiplication triples in the standard model with active, information theoretic security. This construction has a slightly weaker corruption model: we do not allow collusion between clients and servers. Thus, the adversary can corrupt either one client or a minority of the servers.

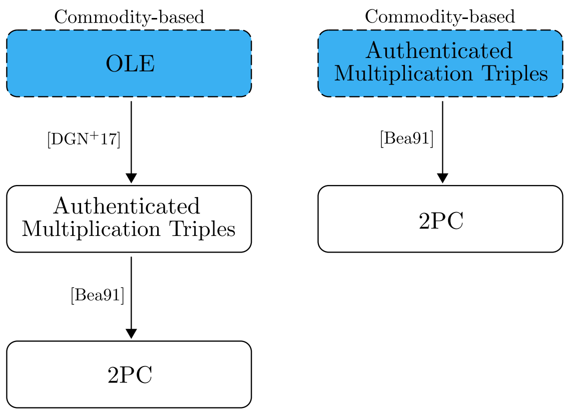
Commodity-based
OLE
[DGN+17]
Authenticated Multiplication Triples
[Bea91]
2PC
Commodity-based
Authenticated Multiplication Triples
[Bea91]
2PC

Fig. 5. An overview of how our constructions (the blue boxes with dashed lines) fits in the bigger picture of performing secure 2-party computation.