Simple Additively Homomorphic UC Commitments from Oblivious Linear Function Evaluation

Zhangxiang Hu* Mike Rosulek*

November 2, 2021

Abstract

An additively homomorphic commitment is one in which the sender can commit to values independently, and later reveal any linear combination of those values to the receiver (i.e., the receiver learns only the result of the linear combination). We describe a very simple additively homomorphic commitment scheme that uses oblivious linear function evaluation (a generalization of oblivious transfer) as a building block. The scheme is UC-secure and has short non-interactive openings.

1 Introduction

An additively homomorphic commitment is one in which the sender can commit to values independently, and later reveal any linear combination of those values to the receiver (i.e., the receiver learns only the result of the linear combination).

An example application of homomorphic commitments is the MiniLEGO protocol of [FJN+13]. There, Alice generates many garbled gates and commits to some appropriate wire labels. Later, some of these commitments are opened (to check the gates) and other *pairs* of commitments are opened to reveal only their XOR (to "solder" the corresponding wires together). The result is a garbled circuit assembled on-the-fly which can then be evaluated, resulting in a general-purpose MPC protocol.

1.1 Our Contributions

We describe an extremely simple *UC-secure, additively homomorphic* commitment scheme based on the building block of *oblivious linear function evaluation* (OLFE) [WW06]. OLFE is a generalization of oblivious transfer, parameterized by a field[1] $GF(q)$. A sender provides inputs $a, b \in GF(q)$ and a receiver provides input $m \in GF(q)$. The sender gets no output while the receiver learns $t = am + b \in GF(q)$. It is easy to see that a single OLFE over $GF(2)$ is equivalent to 1-out-of-2 oblivious transfer of bits.

Another way to look at OLFE is as an oblivious one-time MAC: the sender gives a one-time MAC key (a, b), and the receiver gets a MAC $am + b$ on the message x of their choice. Viewed in this way, it is clear that one instance of OLFE allows the receiver to make a perfectly hiding commitment to m (when a and b are chosen uniformly). The value t can be used as the decommitment; its unforgeability as a one-time MAC directly yields the required binding property (when q is exponential). We note that this aspect of OLFE has been observed before [Riv99].

*Oregon State University, {huz,rosulekm}@eecs.oregonstate.edu. Supported by NSF award CCF-1149647.

[1] The definition makes sense in a ring as well, but our usage of OLFE requires a field.

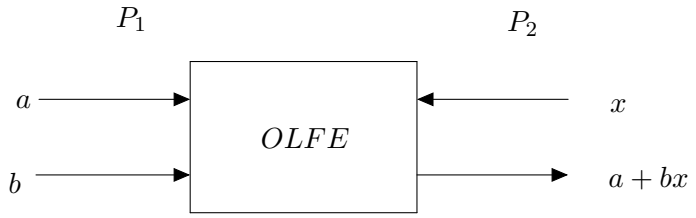

Figure 1: Description of ideal functionality $GF(q)$-OLFE

It is also well-known (e.g., [SW08, AB09, NNOB12]) that MACs of the form $t_i = am_i + b_i$ are additively homomorphic, in the sense that $t_i + t_j$ is also a valid MAC of $m_i + m_j$ under the key $(a, b_i + b_j)$. In our setting, additively homomorphic MACs intuitively yield additively homomorphic commitments.

However, the homomorphic property only holds when the same a is used for each MAC. In our setting, we would require Bob to provide the same a to each instance of OLFE. Indeed, it leads to concrete attacks if Bob sends different a values in some OLFE instances: Imagine Bob has sent different values of a so that Alice receives $t_1 = am_1 + b_1$ and $t_2 = (2a)m_2 + b_2$. Then suppose Alice wishes to decommit to $m_1 + m_2$ by sending $t^* = t_1 + t_2$. Bob can solve for $m_1 + 2m_2 = (t^* - b_1 - b_2)/a$, which is not what Alice agreed to reveal.

Preventing a cheating Bob. Our main challenge, therefore, is to ensure that Bob uses the same a in every OLFE instance. We introduce a very lightweight approach to do so, which guarantees consistency of Bob's a-values, and whose cost is independent of the number of commitments being checked.

Roughly speaking, Alice simply commits to a random value m_0 in addition to her other m_i's. She then sends $t^* = \sum_{i=0}^{n} t_i$ to Bob and challenges him to produce $\sum_{i=0}^{n} m_i$. The random value m_0 is used nowhere else, so this sum does not leak any information about the "real" m_i values. Now, if Bob used the same a value in all of these OLFE instances, one can see that he can easily obtain $\sum m_i = (t^* - \sum b_i)/a$. However, suppose Bob used different a_i values. Then $t^* = \vec{a} \cdot \vec{m} + \vec{b}$, where the vector $\vec{a}$ is linearly independent of the all-ones vector $\vec{1}$. Hence, when the x_i values are randomly distributed, the correct response to Alice's challenge $\vec{1} \cdot \vec{m}$ is distributed independently of Bob's view (which depends only on $\vec{a} \cdot \vec{m}$).

After Alice is convinced that Bob used the same a in each instance of OLFE, she can safely perform non-interactive openings as described above, sending $(\vec{\gamma}, \vec{\gamma} \cdot \vec{m}, \vec{\gamma} \cdot \vec{m})$ to open to the linear combination $\vec{\gamma} \cdot \vec{m}$.

We briefly mention a few subtleties in getting everything to work. First, if Bob uses $a = 0$ in the OLFE involving m_0, then he can indeed learn some information about the m_i's in the challenge-response described above (even though he will eventually get caught). We address this by first performing another challenge-response phase to ensure Bob did not use $a = 0$ for the commitment to m_0. Finally, a cheating Alice can actually learn information about a by sending an incorrect t^* in the challenge-response and observing his reply. We address this by having Alice also commit (using a plain, non-homomorphic commitment) to the correct response in advance. Intuitively, this allows Bob to verify that his response to the challenge doesn't tell Alice anything she didn't already know. We emphasize that this extra step is only needed for the two challenge-response steps; subsequent homomorphic openings are non-interactive.

More exotic openings. Adapting standard tricks from information-theoretic MPC protocols, we show how to extend our homomorphic commitment to allow the sender to reveal a **product** of committed values. That is, Alice can commit to m_1 and m_2, then later open only their product $m_1 m_2$. These multiplicative openings are slightly more expensive and require a slight amount of interaction (i.e., a challenge sent from Bob to Alice).

The homomorphic commitment scheme of [FJN+13] supports additional styles of opening. Suppose both Alice and Bob have some vectors $\vec{\gamma}_0$ and $\vec{\gamma}_1$ in mind. In an **oblivious opening**, Bob provides a bit b and learns the opening $\vec{\gamma}_b \cdot \vec{m}$, while Alice does not learn b. In an **OR-opening**, Alice chooses a bit b and Bob learns $\vec{\gamma}_b \cdot \vec{m}$ but not b. [FJN+13] require these features in their setting, and they achieve them essentially using additively homomorphic commitment as a black-box. We can likewise achieve these features in our scheme.

1.2 Related Work & Comparison

There exist many constructions of commitments that are homomorphic with respect to a group operation (e.g., [Ped91, Gro09]). These have proven to be useful building blocks in the context of efficient NIZK proofs [GOS06, GS08, CKLM12], anonymous credentials (e.g., [CL01]) and elsewhere. These widespread schemes are typically not UC-secure, and commitments in these scheme consist of group elements in cryptographically hard groups. Then one can commit to either discrete logarithms of group elements, or to group elements themselves.

Unlike these schemes, our work follows in the setting of [FJN+13, DDGN14]. These schemes achieve UC security and involve commitments of bit-strings (with XOR-homomorphism) or finite field elements, hence the protocol gains its security from somewhere other than the choice of algebraic group/field. Regarding use of expensive public-key machinery, this has been shown to be in some sense necessary for UC-secure (not necessarily homomorphic) commitments [DNO10]. One can see that all of these schemes inherently rely on a complete setup primitive such as oblivious transfer or OLFE in our case. Yet, these schemes (and our own) do not use any public-key machinery during the opening phase.

In [DDGN14], much of the efficiency is gained by using packed secret sharing, which allows some of the overhead to be amortized across several group elements. Concretely, in their proposed instantiation Alice commits to a vector of field elements at a time. Alice can easily open a linear combination of the form $\sum_i \gamma_i \boldsymbol{m}_i$, where each $\boldsymbol{m}_i$ is itself a vector and γ_i is a scalar. However, this batching of field elements does come at a cost of flexibility. Consider a setting in which Alice has committed to many field elements $\{m_i\}$ and would like to open $m_i + m_j$, for two arbitrary indices i and j (this is the use case of homomorphic commitment in the setting of [FJN+13], where the field elements represent wire labels). To do this using the packed-secret-share instantiation of [DDGN14] is cumbersome. First, one must apply a linear operator to the vector containing element m_i to "get m_i out" of that vector. This involves giving an additional commitment to $\boldsymbol{m}' = \phi(\boldsymbol{m})$, where ϕ is an appropriate linear operator, then performing an *interactive* procedure to prove that the two commitments are related according to ϕ, and only then open (a linear combination involving) $\boldsymbol{m}'$. Furthermore, the setup phase is linear in the number of distinct ϕ's that will be supported for this kind of opening. Hence, in settings where field elements are not naturally grouped into long vectors, many of the benefits of the [DDGN14] protocol are less clear.

In [FJN+13], the protocol supports only XOR homomorphism, and not more general linear operations in $GF(2^k)$ as in our scheme. The functionality they achieve, though sufficient for their setting, is also leaky in some sense. They cannot guarantee that every commitment is binding — a malicious Alice can equivocate to some degree on up to k commitments, where k is the security parameter. Again, this is sufficient for their setting, where their use of the functionality accounts for

these "wildcard" commitments, but it does not appear straight-forward to "repair" their protocol to achieve full security to be most useful in wider settings.

The previous two protocols [FJN+13, DDGN14] are based on k-out-of-n oblivious transfer, whereas ours is based on the less standard OLFE. Ishai, Prabhakaran, and Sahai [IPS09] describe a general protocol construction for arithmetic circuits in the OT-hybrid model with low amortized overhead. Since OLFE can be represented by an arithmetic circuit with just 2 gates, their construction yields an OLFE protocol with (amortized) constant number of field elements communicated per OLFE and computation roughly $O(\log k)$ field operations per OLFE.

As described earlier, our protocol uses the homomorphic properties of the simple algebraic MAC ax_i+b_i. This homomorphic MAC has proven useful in many settings, including proofs of retrievability [SW08], authenticated network coding [AB09], and secure multiparty computation [NNOB12], to name a few.

2 Preliminaries

We use $[n]$ to denote the set $\{1, \ldots, n\}$. We use $GF(q)$ to denote a finite field of q elements (where q is a prime power). We call a function ν *negligible* if for all $c > 0$, $\nu(n) < 1/n^c$ for all but finitely many n.

Commitment Commitment scheme plays a important role in cryptography, such as in zero-knowledge proof and secure computation. A commitment scheme is a protocol between two parties P_1 and P_2. It allows one party P_1 to commit to a secret value m while leaking no information about m. Also, P_1 can reveal this value to P_2 later.

A commitment scheme requires two intuitive properties: privacy and binding. Privacy means P_2 learns nothing about the secret value m during the commit phase. Binding requires that P_2 learns the secret m during the reveal phase and m is the only value that P_2 will accept.

In our protocol, we will use a plain, non-homomorphic commitment $\mathcal{F}_{\mathsf{com}}$ as an ideal functionality. On input $(\textsc{commit}, m)$ from the sender, $\mathcal{F}_{\mathsf{com}}$ stores m and sends $(\textsc{committed}, m)$ to the receiver. Then in the reveal phase, on input $(\textsc{open}, m)$ from the sender, $\mathcal{F}_{\mathsf{com}}$ outputs $(\textsc{opened}, m)$ to the receiver.

Homomorphic commitment A more powerful commtment scheme is homomorphic commitment scheme. A homomorphic commitment scheme is similar as a general commitment scheme except that homomorphic commitment allows P_1 to commit to a bunch of values $(m_1, \cdots m_n)$, and P_1 can open a combination of such messages without leaking any other information. There are many existing schemes that acheive different kind of homomorphic commitments such as XOR-homomorphism, additively homomorphism and multiplicative homomorphism.

In this paper, we will focus on an additively homomorphic commitment. Our protocol allows P_1 to open any single value or any linear combinations of such committed values. To implement our protocol, we will need to use a very useful primitive: Oblivious Linear-Function Evaluation.

Oblivious Linear-Function Evaluation The most important primitive that we use in our homomorphic commitment scheme is *oblivious linear function evaluation over* $GF(q)$ ($GF(q)$-OLFE). It is introduced in [WW06]. We let $\mathcal{F}^q_{\mathsf{OLFE}}$ denote the ideal functionality for $GF(q)$-OLFE. The sender P_1 gives inputs $a, b \in GF(q)$, and the receiver P_2 gives input $x \in GF(q)$ and learns the output $y = ax + b$ while P_1 gets no output. A special case is $GF(2)$-OLFE, which is equivalent to 1-out-of-2 oblivious transfer of bits. The description of $\mathcal{F}^q_{\mathsf{OLFE}}$ is showed in Figure 1.

3 Homomorphic Commitment

We now give the formal description of our additively homomorphic commitment protocol. In both the ideal functionality and protocol, we let P_1 to be the sender and P_2 the receiver.

3.1 Ideal functionality

$\mathcal{F}_{\mathsf{Hcom}}^{q}$ describes the ideal functionality for our homomorphic commitment scheme. Here q is a global parameter that both parties agrees on, which determines the field $GF(q)$ over which our scheme is homomorphic. For technical reasons, $\mathcal{F}_{\mathsf{Hcom}}^{q}$ maintains a set C of message indices. Only messages in C can be opened in an opening step, and P_1 must explicitly ask messages to be added to C.

$\mathcal{F}_{\mathsf{Hcom}}^{q}$ allows sender P_1 to commit to messages and later to open any linear combination of committed messages without revealing any other information about the original messages. The full description of $\mathcal{F}_{\mathsf{Hcom}}^{q}$ is defined in Figure 2. It consists of SETUP, COMMIT, CHECK and OPENING steps.

The functionality is parameterized by a prime power $q > 2^k$.

Setup: $\mathcal{F}_{\mathsf{Hcom}}^{q}$ creates an initially empty vector $\vec{m}$ of messages, an integer variable $N = 0$, and an initially empty subset C.

Commit: On input (COMMIT, m) from P_1, where $m \in GF(q)$: Set $\vec{m} = (\vec{m} \| m)$ and set $N = N + 1$. Then send $(\text{COMMITTED}, N)$ to P_2.

Check: On input $(\text{CHECK}, \mathcal{M})$ from P_1, where $\mathcal{M} \subseteq [N]$, send $(\text{CHECK}, \mathcal{M})$ to P_2. If P_2 is honest, or if corrupt P_2 sends CONFIRM to $\mathcal{F}_{\mathsf{Hcom}}^{q}$, then set $C = C \cup \mathcal{M}$ send CHECKED to P_1. Otherwise if P_2 sends ABORT, send $\perp$ to P_1 and stop responding.

Opening: On input $(\text{OPEN}, \vec{\gamma})$ from P_1, where $\vec{\gamma} \in GF(q)^N$: If there is an $i \in [N] \setminus C$ such that $\gamma_i \neq 0$, then abort and send $\perp$ to P_2. Otherwise, send $(\text{OPENED}, \vec{\gamma}, \vec{\gamma} \cdot \vec{m})$ to P_2.

Figure 2: Description of ideal functionality $\mathcal{F}_{\mathsf{Hcom}}^{q}$

3.2 Our Protocol

In this section, we give a formal description of the protocol π which securely realizes $\mathcal{F}_{\mathsf{Hcom}}^{q}$. It consists four steps: **Setup**, **Commit**, **Check** and **Opening**. Our protocol uses ideal setup functionalities $\mathcal{F}_{\mathsf{OLFE}}^{q}$ for OLFE and $\mathcal{F}_{\mathsf{com}}$ for (plain) commitment. Both parties will setup a global parameter q at the begining such that all inputs to the protocol should be the elements of $GF(q)$.

For simplicity, the protocol requires P_1 to commit to all messages in one phase before performing any openings. Later, we describe how P_1 can interleave commits and openings.

The protocol is as follows:

- **Setup:** Both parties agree on the global parameter q. P_1 initialize vectors $\vec{t} = \emptyset$, $\vec{m} = \emptyset$. P_2 picks a random $a \in_R GF(q) \setminus \{0\}$ and initialize vector $\vec{b} = \emptyset$. Both parties set $N = 0$.

  P_1 generate two random variables $r_0 \in_R GF(q)$ and $r_1 \in_R GF(q)$, P_2 generates $b_{r_0} \in_R GF(q)$ and $b_{r_1} \in_R GF(q)$. P_1 and P_2 invoke $\mathcal{F}_{\mathsf{OLFE}}^{q}$ and P_1 receives $t_{r_0} = ar_0 + b_{r_0}$, $t_{r_1} = ar_1 + b_{r_1}$.

- **Commit:** When P_1 receives input (COMMIT, m), P_1 set $m_i = m$ and use it as the input to $\mathcal{F}_{\mathsf{OLFE}}^q$. P_2 chooses a random $b_i \in_R GF(q)$, takes a and b_i as the input to $\mathcal{F}_{\mathsf{OLFE}}^q$. P_1 receives $t_i = am_i + b_i$ from $\mathcal{F}_{\mathsf{OLFE}}^q$. P_1 updates $\vec{t} = (\vec{t} \| t_i)$ and $\vec{m} = (\vec{m} \| m_i)$. P_2 updates $\vec{b} = (\vec{b} \| b_i)$ and Both partites set $N = N + 1$. P_2 outputs $(\text{COMMITTED}, N)$.

- **Check:** In this step, the protocol checks if all a_i are equal. When P_1 receives input $(\text{CHECK}, \mathcal{M})$, do the following:

  - Step A: open $r_0 + r_1$
    1. P_1 solves $t_r = t_{r_0} + t_{r_1}$ and $r^* = r_0 + r_1$. P_1 sends t_r as well as a plain-commitment $\mathcal{F}_{\mathsf{com}}$ of r^* to P_2.
    2. P_2 receives t_r and solves $\tilde{r} = \frac{t_r - (b_{r_0} + b_{r_1})}{a}$. P_2 sends $\tilde{r}$ to P_1.
    3. P_1 checks if $\tilde{r} = r^*$. If so, sends (OPEN) to the instance of $\mathcal{F}_{\mathsf{com}}$ and P_2 receives r^*. Otherwise outputs $\perp$ and aborts the protocol.
    4. P_2 checks if $r^* = \tilde{r}$. If not, P_2 outputs $\perp$ and aborts the protocol.
  - Step B: open $\sum_{i=1}^{N} m_i + r_1$
    1. P_1 solves $t^* = \sum_{i=1}^{N} t_i + t_{r_1}$ and $m^* = \sum_{i=1}^{N} m_i + r_1$. P_1 sends t^* as well as a plain-commitment $\mathcal{F}_{\mathsf{com}}$ of m^* to P_2
    2. P_2 receives t^* and solves $\tilde{m} = \frac{t^* - (\sum_{i=1}^{N} b_i + b_{r_1})}{a}$. P_2 sends $\tilde{m}$ to P_1.
    3. P_1 checks if $\tilde{m} = m^*$. If so, opens the $\mathcal{F}_{\mathsf{com}}$-commitment and P_2 receives m^*. Otherwise outputs $\perp$ and aborts the protocol.
    4. P_2 checks if $m^* = \tilde{m}$. If not, P_2 outputs $\perp$ and aborts the protocol.

- **Opening:** When P_1 receives input $(\text{OPEN}, \vec{\gamma})$, to implement homomorphic opening, do the following:

  1. P_1 computes $T^* = \vec{\gamma} \cdot \vec{t}$ and $M^* = \vec{\gamma} \cdot \vec{m}$, then sends $(\vec{\gamma}, M^*, T^*)$ to P_2.
  2. P_2 computes $\tilde{M} = \frac{T^* - \vec{\gamma} \cdot \vec{b}}{a}$, if $\tilde{M} = M^*$, P_2 outputs $(\text{OPENED}, \vec{\gamma}, M^*)$. Otherwise P_2 outputs $\perp$.

As motivated in the introduction, the purpose of the **Check** steps is to detect whether P_2 has used different values of a as inputs to $\mathcal{F}_{\mathsf{OLFE}}^q$. In section 3.3, we will show that when P_2 is cheating, he fails the **Check** steps with overwhelming probability. The idea behind this is that, in step A P_1 aborts unless $a_{r_0} = a_{r_1}$. Then in step B P_1 aborts unless all a_i are equal. If P_1 aborts in step A, P_2 only learns information about r_1 and r_2. If $P1$ aborts in step B, P_2 also learns nothing about the original message due to the mask value. After the **Check** steps are complete, a_i are guaranteed to be equal which means P_2 is not cheating, the protocol can start the normal (less expensive) homomorphic opening. If both parties are honest, then the messages can be opened correctly in **Opening** step.

3.3 Security proof

We now prove the security of our protocol.

Theorem 1. *The protocol π is a UC-secure realization of $\mathcal{F}_{\mathsf{Hcom}}^q$ when P_1 chooses each message m_i uniformly random from $GF(q)$.*

Proof. Looking ahead, the reason that we require m_i is uniformly random is that the randomness in m_i is used to help P_1 detect cheating behavior of P_2.

Case 1: P_2 is corrupt. We construct a simulator for corrupt P_2 in a sequence of hybrids:

$\mathcal{H}_0$: We define $\mathcal{H}_0$ to be identical to the real protocol as in Section 3.2. The simulator $\mathcal{S}$ uses P_1's input to run the protocol on P_1's behalf. $\mathcal{S}$ also faithfully simulates the ideal $\mathcal{F}_{\mathsf{OLFE}}^q$ and (plain) commitment functionalities. One thing we highlight here is that $\mathcal{S}$ can extract P_2's input from $\mathcal{F}_{\mathsf{OLFE}}^q$. First $\mathcal{S}$ initialize vectors $\vec{a}$ and $\vec{b}$. For the ith invocation of $\mathcal{F}_{\mathsf{OLFE}}^q$, $\mathcal{S}$ extracts a_i and b_i, then updates $\vec{a} = (\vec{a}\|a_i)$ and $\vec{b} = (\vec{b}\|b_i)$. We summarize the view of P_2 in our protocol:

1. **Setup**. P_2 receives nothing
2. **Commit**. P_2 does not receive anything in this part. Both parties just agree on a global parameter q and P_2 outputs (COMMITTED, N).
3. **Check**.
   - Step A: P_2 receives $t_r = t_{r_0} + t_{r_1}$, a plain-commitment $\mathcal{F}_{\mathsf{com}}$ of $r^* = r_0 + r_1$ and an open instance of such $\mathcal{F}_{\mathsf{com}}$.
   - Step B: P_2 receives $t^* = \sum_{i=1}^{N} t_i + t_{r_1}$, a plain-commitment $\mathcal{F}_{\mathsf{com}}$ of $m^* = \sum_{i=1}^{N} m_i + r_1$ and an open instance of such $\mathcal{F}_{\mathsf{com}}$.
4. **Opening**. P_2 receives $T^* = \vec{\gamma} \cdot \vec{t}$ and $M^* = \vec{\gamma} \cdot \vec{m}$

There is nothing in P_2's view of **Commit** and **Setup** steps, $\mathcal{S}$ just act the same as a honest P_1. Agree on the global parameter q, keep track on the value of N, act as $\mathcal{F}_{\mathsf{OLFE}}^q$ and create two vectors $\vec{a}$, $\vec{b}$ to store corresponding (a_i, b_i).

$\mathcal{H}_1$: We define $\mathcal{H}_1$ the same as $\mathcal{H}_0$ except for the following: after checking the condition $\tilde{r} = r^*$, the simulator will also abort if $(a_{r_0} \neq a_{r_1}) \vee a_{r_0} = 0 \vee a_{r_1} = 0$. Notice that in the ideal world, $\mathcal{S}$ can extract a_{r_0} and a_{r_1} as P_2's inputs to $\mathcal{F}_{\mathsf{OLFE}}^q$.

Hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ differ only in the event that P_2 sends correct $\tilde{r} = r^*$ and yet $a_{r_0} \neq a_{r_1}$ or some of $\{a_{r_0}, a_{r_1}\}$ are zero. To show that the hybrids are indistinguishable, it suffices to prove that this event happens only with negligible probability.

To see why, assume P_2 picks $a_{r_0} \neq a_{r_1}$ and sends correct $\tilde{r}$. In the protocol P_2 receives $t_r = a_{r_0} r_0 + a_{r_1} r_1 + (b_{r_0} + b_{r_1})$. In other words, it depends only on $a_{r_0} r_0 + a_{r_1} r_1$. When r_0 and r_1 are chosen uniformly, $a_{r_0} r_0 + a_{r_1} r_1$ is distributed independently of the "correct answer" $r^* = r_0 + r_1$. Hence, P_2 achieves $\tilde{r} = r^*$ with probability at most $1/q$ which is negligible.

Note that when P_2 chooses non-zero $a_{r_0} = a_{r_1}$, P_2 can still send an incorrect $\tilde{r} \neq r^*$. Hence the simulator must still check the condition $\tilde{r} = r^*$, even in the case $a_{r_0} = a_{r_1}$.

$\mathcal{H}_2$: Next, we construct $\mathcal{S}$ by changing the view of P_2 in Step B. We define $\mathcal{H}_2$ to be identical to $\mathcal{H}_1$ except that $\mathcal{S}$ chooses random $t_s^* \in_R GF(q)$ in Step B and simulates a commitment. Then, when P_2 responds with $\tilde{m}$, we abort if not all $\{a_i \mid i \in [N]\}$ are equal to a_{r_1}. If all are equal, then we set $m^* = \frac{t_s^* - (\sum_{i=1}^{N} b_i + b_{r_1})}{a_{r_1}}$ and proceed to check $m^* = \tilde{m}$ as usual.

In $\mathcal{H}_1$, the value t^* is computed as $t^* = \sum_{i=1}^{N} t_i + t_{r_1} = \sum_{i=1}^{N} t_i + (a_{r_1} r_1 + b_{r_1})$. From the above, we have that $a_{r_1} \neq 0$ and that $\mathcal{S}$'s view is independent of r_1. So t^* is uniform in both $\mathcal{H}_1$ and $\mathcal{H}_2$.

In the case that all $a_{r_1} = a_i$ for all i, it is easy to see that (m^*, t^*) have the same joint distribution in both $\mathcal{H}_1$ and $\mathcal{H}_2$.

So the two hybrids differ only when $\{a_i \mid i \in [N]\}$ are not all equal to a_{r_1} and P_2 can correctly produce $\tilde{m} = m^*$. By the same reasoning as above, this happens with only negligible probability $1/q$. Hence the two hybrids are indistinguishable.

$\mathcal{H}_3$: For the last hybrid, we simulate the P_2's view in *Opening*. If $\mathcal{S}$ has not aborted, we have that all a_i values are identical. Let use use a to denote their common value.

We define $\mathcal{H}_3$ the same as $\mathcal{H}_2$ except for the following. Instead of computing $T^* = \vec{\gamma} \cdot \vec{t}$, we use $T^* = a(\vec{\gamma} \cdot \vec{m}) + \vec{\gamma} \cdot \vec{b}$. It is clear that the two expressions are equal, so the hybrids are identical.

Now the simulator $\mathcal{S}$ in $\mathcal{H}_3$ does not need any information beyond what P_2 learns from $\mathcal{F}^q_{\mathsf{Hcom}}$ in the ideal world. In particular, it does not use $\vec{m}$ values in the Setup and Check steps. In the opening steps, it uses only $\vec{\gamma}$ and $\vec{\gamma} \cdot \vec{m}$ from the ideal functionality.

Case 2: P_1 is corrupt. Again we construct the simulator for corrupt P_1 in a series of hybrids.

$\mathcal{H}_0$: We define $\mathcal{H}_0$ to be the real protocol as in Section 3.2. The simulator $\mathcal{S}$ runs the protocol honestly on P_2's behalf. $\mathcal{S}$ also faithfully simulates the ideal $\mathcal{F}^q_{\mathsf{OLFE}}$ and (plain) commitment functionalities. Notice that $\mathcal{S}$ can extract P_1's input from $\mathcal{F}^q_{\mathsf{OLFE}}$. $\mathcal{S}$ initialize a vector $\vec{m}$. For the ith invocation of $\mathcal{F}^q_{\mathsf{OLFE}}$ in **Commit**, $\mathcal{S}$ receives m_i and updates $\vec{m} = (\vec{m} \| m_i)$. Also, $\mathcal{S}$ receives r_0 and r_1 in **Setup** from $\mathcal{F}^q_{\mathsf{OLFE}}$.

Consider P_1's view during the protocol, it consists of:

1. **Setup**. There are two instances of $\mathcal{F}^q_{\mathsf{OLFE}}$ invocation, P_1 receives t_{r_0} and t_{r_1}
2. **Commit**. For each invocation of $\mathcal{F}^q_{\mathsf{OLFE}}$, P_1 receives corresponding t_i.
3. **Check**.
   - Step A: P_1 receives $\tilde{r}$
   - Step B: P_1 receives $\tilde{m}$
4. **Opening**. P_1 does not receive anything in this step.

Now we need simulate all things we described above and also we need to induce correct outputs (including abort) for P_2. First we let $\mathcal{S}$ act as an honest P_2: agree on the global parameter q, initialize N and updates it when invokes $\mathcal{F}^q_{\mathsf{OLFE}}$ on message m_i. Also, $\mathcal{S}$ initialize a vector $\vec{m}$ to store messages. Next, we describe the sequence of hybrids to construct the simulator $\mathcal{S}$.

$\mathcal{H}_1$: We can see that in **Commit** step, P_1 receives t_i for each invocation of $\mathcal{F}^q_{\mathsf{OLFE}}$. In **Setup**, P_1 receives t_{r_0} and t_{r_1}.

In $\mathcal{H}_0$ these are computed by first picking random b_i and setting $t_i = am_i + b_i$. In $\mathcal{H}_1$, we instead first choose random t_i and then set $b_i = t_i - am_i$ accordingly. In this way, the t_i values sent to P_2 are distributed independently of a. It is not hard to see that $\mathcal{H}_1$ and $\mathcal{H}_0$ have the same distribution. Then we have that $\mathcal{H}_1 \equiv \mathcal{H}_0$. Also remember that $\mathcal{S}$ can extract P_1's inputs m_i, r_0 and r_1 from the ideal functionality $\mathcal{F}^q_{\mathsf{OLFE}}$.

$\mathcal{H}_2$: In step A of **Check**, we not only need to simulate $\tilde{r}$, also the output and abort probability of P_2. We define $\mathcal{H}_2$ identical to $\mathcal{H}_1$ except that after checking $r^* = \tilde{r}$ at the end of Step A, $\mathcal{S}$ also aborts if $r^* \neq r_0 + r_1$, where r_0 and r_1 were extracted from $\mathcal{F}^q_{\mathsf{OLFE}}$.

To see why the two hybrids are indistinguishable, suppose that P_1 commits to $r^* = r_0 + r_1 + \delta$. In order for P_2 not to abort, P_1 needs to send a corresponding t'_r satisfying the following:

$$\begin{aligned} t'_r &= a(r_0 + r_1 + \delta) + b_{r_0} + b_{r_1} \\ &= (t_{r_0} + t_{r_1}) + a\delta \end{aligned}$$

P_1 knows t_{r_0} and t_{r_1} as well as δ, hence in this case P_1 can compute the correct a when $\delta \neq 0$. However, from above we know that P_1's view is independent of a. Hence, the event on which $\mathcal{H}_1$ and $\mathcal{H}_2$ differ (which corresponds to $\delta \neq 0$) can happen only with probability at most $1/q$, which is negligible.

$\mathcal{H}_3$: Next we change the behavior of $\mathcal{S}$ in step B of **Check** step. We define $\mathcal{H}_3$ to be the same as $\mathcal{H}_2$ except that, after checking the condition $m^* = \tilde{m}$, $\mathcal{S}$ additionally aborts if $m^* \neq \sum_{i=1}^{N} m_i + r_1$.

From the same argument as above, if P_1 can send an incorrect m^* along with a valid t, then P_1 could also produce a. Yet P_1's view is is independent of a. Therefore, we have that $\mathcal{H}_3 \approx \mathcal{H}_2$.

$\mathcal{H}_4$: Since P_1 does not receive anything in **Opening**, we only need to consider the outputs and abort probability of P_2. We define $\mathcal{H}_4$ to be identical to $\mathcal{H}_3$ except that, after checking the condition $\tilde{M} = M^*$, $\mathcal{S}$ also aborts if $M^* \neq \vec{\gamma} \cdot \vec{m}$, where $\vec{m}$ are P_1's extracted inputs to $\mathcal{F}^q_{\mathsf{OLFE}}$. Again, following the same argument as before, we have that $\mathcal{H}_4 \approx \mathcal{H}_3$.

$\mathcal{H}_5$: We let $\mathcal{H}_5$ be the same as $\mathcal{H}_4$, with the following changes. Each time P_1 gives an input m_i to $\mathcal{F}^q_{\mathsf{OLFE}}$, $\mathcal{S}$ sends (COMMIT, m_i) to $\mathcal{F}^q_{\mathsf{Hcom}}$. When P_1 performs the **Check** step and the simulated P_2 does not abort, $\mathcal{S}$ sends $(\text{CHECK}, \{m_i\})$ to $\mathcal{F}^q_{\mathsf{Hcom}}$. Each time P_1 opens a commitment and the simulated P_2 does not abort, $\mathcal{S}$ sends $(\text{OPEN}, \vec{\gamma})$ to $\mathcal{F}^q_{\mathsf{Hcom}}$.

From the additional checks introduced in our hybrids, we have in $\mathcal{H}_5$ that the outputs of the simulated P_2 and ideal P_2 always match. This $\mathcal{S}$ defines our final simulator for corrupt P_1. □

4 Extensions

4.1 Interleaving Commits and Openings, Dealing with Non-Random Messages

As we have mentioned above, protocol π requires all messages to be committed before any openings are performed. Now we describe how to interleave commits and openings arbitrarily.

The idea is simple and we only need a little extra cost. Recall that in the **Check** step of the protocol, P_1 uses $\mathcal{F}^q_{\mathsf{OLFE}}$ to commit to random values r_1, r_2 which are used to test for a cheating P_2. Let us use *cautious opening* to refer to the interactive challenge-response method by which P_1 opens the sums $r_1 + r_2$ and $r_1 + \sum_i m_i$. The property of a cautious opening is that if P_2 cheats (i.e., uses inconsistent values of a in the OLFE instances for those commitments), then P_1 catches P_2. In other words, if a cautious opening succeeds, then P_1 is assured that P_2 used consistent a values for each of the values used in the cautious opening.

To modify our protocol, we have P_1 commit to an initial random value r_0 and defer the commitments to r_1, r_2. Now, consider the jth time that P_1 has a $(\text{CHECK}, \mathcal{M})$ command (recall that P_1 must check each commitment before it is used in an opening). For this jth check command, P_1 commits to random values r_{2j-1}, r_{2j} and does a cautious opening of $r_{2j-2} + r_{2j-1} + r_{2j}$. If it

succeeds, then by induction P_2 must have used the same nonzero a value for all commitments to r-values. Then P_1 can do a cautious opening of $r_{2j-1} + \sum_{i \in \mathcal{M}} m_i$. Since P_2 used a nonzero a value for the commitment to r_{2j-1}, even a cheating P_2 learns nothing about $\vec{m}$ from this cautious opening.

Note that we only need the property that, at the time a command $(\text{CHECK}, \mathcal{M})$ is executed, the values in $\{m_i \mid i \in \mathcal{M}\}$ are uniformly, independently distributed given P_2's view. In particular, they need not be independent of other previously committed messages, if those messages have not yet been opened.

4.2 Achieving Multiplicative Homomorphism

Using relatively standard techniques from secret-sharing-based MPC (adapted from the presentation in [CDN12]), we can extend our commitment scheme to be multiplicatively homomorphic.

Suppose P_1 has committed to values a and b and would like to open their product ab. The protocol is as follows:

1. P_1 chooses random $r \leftarrow GF(q)$ and sets $z = rb$, $c = ab$. P_1 makes homomorphic commitments to r and to z, and makes a plain commitment to c.
2. P_2 chooses a random challenge $e \leftarrow GF(q)$ and sends it to P_1.
3. P_1 computes $X = ea + r$ and $Y = Xb - z$, then performs homomorphic openings to X (as a public linear combination of a and r) and Y (as a public linear combination of b and z). P_1 also opens the plain commitment to c.
4. P_2 checks that the correct X was used in the linear combination that defined the homomorphic opening of Y. Then P_2 checks that $Y = ec$, aborting if this is not the case.

To see why this is secure against a cheating P_1, consider that at step 1, P_1 is committed to both c and z. Suppose that P_1 has committed to $c = ab + \Delta_c$ and also $z = rb + \Delta_z$ (i.e., if these Δ values are non-zero, then P_1 is not following the protocol). P_2 finally checks the equality:

$$\begin{aligned} & ec = Y \\ \iff & e(ab + \Delta_c) = Xb - z \\ & \quad = (ea + r)b - z = eab + rb - (rb + \Delta_z) \\ \iff & e\Delta_c = -\Delta_z \end{aligned}$$

Now, e was chosen uniformly after Δ_c and Δ_z were fixed. Hence, if $(\Delta_c, \Delta_z) \neq (0, 0)$, then equality holds with at most negligible probability $1/q$.

To see why this is secure against a cheating P_2, observe that P_2 learns only X, Y, and c. Since $X = ea + r$ and r is chosen randomly by P_1, X is uniform as well. Then Y is the unique value such that $ec = Y$. Hence, P_2's view can be simulated knowing only $c = ab$.

Now, it is true that the values r and z are correlated with b. Hence, P_1 does not commit to totally independently uniform messages. However, in the case that b has not been used in any opening, then the marginal distribution of (r, z) is indeed uniform (since b was uniform). Hence, in these cases the multiplicative opening can be carried out using the interleaved commit/open technique described above.

4.3 Oblivious Openings

In [FJN+13], several additional styles of openings were required for their usage of homomorphic commitment. Our protocol can be augmented using identical techniques as [FJN+13] to achieve these additional features. For completeness, we sketch how to implement those features.

Suppose both P_1 and P_2 agree on two vectors $\vec{\gamma}_0$ and $\vec{\gamma}_1$. In both of these special openings, P_2 will eventually learn an opening of $\vec{\gamma}_b \cdot \vec{m}$ for some bit b.

- In an **oblivious opening**, P_2 chooses the bit b and P_1 does not learn it.
- In an **OR-opening**, P_1 chooses the bit b and P_2 does not learn it.

For an oblivious opening, [FJN+13] take advantage of the fact that openings are non-interactive. Hence, P_1 can use the openings of $\vec{\gamma}_0 \cdot \vec{m}$ and $\vec{\gamma}_1 \cdot \vec{m}$ as inputs to an oblivious transfer, with P_2 using bit b to select the appropriate one. This introduces a selective-abort attack for P_1, by replacing one of these openings with a junk value. If P_2 aborts, then P_1 learns what his bit b must have been. In their setting, this selective abort is not problematic, and they fold it into the definition of the ideal functionality. We observe that our scheme has non-interactive openings as well, and the same approach can be used here.

For an OR-opening, we have P_1 choose a random secret bit p and commit to $m_0^* = \vec{\gamma}_p \cdot \vec{m}$ and $m_1^* = \vec{\gamma}_{1-p} \cdot \vec{m}$. Then P_2 sends a random challenge bit $c = 0$. If $c = 0$ then P_1 reveals p and proves that the commitments to m_0^*, m_1^* are consistent. For this, he opens the linear combination $\vec{\gamma}_b \cdot \vec{m} - m_p^*$ for $b \in \{0, 1\}$. The result for each opening will be zero. If $c = 1$ then P_1 simply opens $m_{b \oplus p}^*$. This is a zero-knowledge proof with soundness error $\frac{1}{2}$, which can be repeated k times. We point out that the protocol of [FJN+13] admits "wildcard" commitments on which a cheating P_1 can equivocate. Hence, slightly more tricks are needed for an OR-opening. However, the simpler technique described here works for our protocol since there are no wildcard commitments.

References

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 292–305. Springer, June 2009.

[CDN12] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Secure multiparty computation and secret sharing – an information theoretic approach, 2012. Book draft. Retrieved online from http://cs.au.dk/~jbn/mpc-book.pdf.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300. Springer, April 2012.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, May 2001.

[DDGN14] Ivan Damgård, Bernardo David, Irene Giacomelli, and Jesper Buus Nielsen. Compact VSS and efficient homomorphic UC commitments. Cryptology ePrint Archive, Report 2014/370, 2014. http://eprint.iacr.org/2014/370.

[DNO10] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. On the necessary and sufficient assumptions for UC computation. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 109–127. Springer, February 2010.

[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, May 2013.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, August 2006.

[Gro09] Jens Groth. Homomorphic trapdoor commitments to group elements. Cryptology ePrint Archive, Report 2009/007, 2009. http://eprint.iacr.org/2009/007.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, April 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, March 2009.

[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, August 2012.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, August 1991.

[Riv99] Ronald L. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer, 1999. Unpublished manuscript. Retrieved online from http://people.csail.mit.edu/rivest/Rivest-commitment.pdf.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 90–107. Springer, December 2008.

[WW06] Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 222–232. Springer, May / June 2006.