Lattices Zero Knowledge Proofs from “MPC-in-the-head”

September 13, 2018

1 Introduction

Let q be a prime and $n, m \in \mathbb{N}, n < m$. $\mathbb{Z}_q$ is the ring obtained via modular reduction. Let $\lambda \in \mathbb{N}$ be the computational security parameter and $\kappa \in \mathbb{N}$ be the statistical security parameter. Next, let $S_\beta^m \subset \mathbb{Z}^m$ be a subset of m-elements vectors with ℓ_∞-norm $\leq \beta$. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be a matrix and $\vec{s} \in S_\beta^m, \vec{t} = \mathbf{A} \cdot \vec{s} \bmod q$. For this work, we take $\beta = 1$.

In this document, we describe a protocol for proving knowledge of $s \in S_1^m$ such that $\vec{t} = \mathbf{A}\vec{s}$ for given $\vec{t}$ and $\mathbf{A}$. We will use the “MPC-in-the-head” approach, where the prover commits to a transcript of an MPC protocol computing a circuit which represents the argument that is being proved (and outputs 1 in case that the argument holds and 0 otherwise). Upon receiving a random challenge from the verifier, the prover opens the view of a subset of the parties during the execution. Then, the verifier checks that the opened views are consistent and that the output of the parties is 1. If the conditions hold then the parties output accept.

1.1 The Circuit

We start by defining the circuit C that is being computed. Given the public matrix $\mathbf{A} = (a_{k,\ell})$ and vector $\vec{t} = (t_\ell)$, the circuit outputs 1 if and only if:

1. $\forall \ell \in \{1, \dots, n\} : \ t_\ell = \sum_{k=1}^m a_{\ell,k} s_k$
2. $\forall k \in \{1 \dots, m\} : \ s_k \in \{0, 1\}$

where $s_1, \dots, s_m$ is the secret input to the circuit (known only to the prover in our protocol).

The above is equivalent to saying that the circuit outputs 1 if and only if:

1. $\forall \ell \in \{1, \dots, n\} : \ t_\ell - \sum_{k=1}^m a_{\ell,k} s_k = 0$
2. $\forall k \in \{1 \dots, m\} : \ s_k(s_k - 1) = 0$

We say that $C(\vec{s}) = 1$ iff the above conditions hold for an input $\vec{s}$.

Observe that the first condition contains only linear operations (multiplication by a constant and addition), whereas the second condition can be written as $s_k^2 - s_k = 0$, thus containing a square operation and an addition.

1.2 The MPC protocol

Let N denote the number of parties and let $P_1, \dots, P_N$ denote the parties participating in the protocol.

Secret sharing scheme. Let $[\![x]\!]$ denote the a sharing of x. We use a simple additive secret sharing with the following operations:

- $\mathsf{share}(x)$: A procedure where the dealer who holds a value x, chooses random $x_1, \ldots, x_n \in \mathbb{Z}_q$ such that $x = x_1 + \cdots x_n \bmod q$ and sends x_i to P_i.
- $\mathsf{open}([\![x]\!])$: In this procedure, the parties reveal the secret x by having each party sending each share. Upon receiving x_j from each P_j, party P_i computes $x = \sum_{j=1}^{N} x_j \bmod q$.
- $[\![x]\!] + [\![y]\!]$: Given two shares x_i and y_i of x and y, each party P_i defines $x_i + y_i$ as its share of the result.

Square operation. We say that the pair $([\![b]\!], [\![b^2]\!])$ is a random square if b is random. To compute the square $[\![x^2]\!]$ given $[\![x]\!]$ using a preprocessed $([\![b]\!], [\![b^2]\!])$, the parties work as follows:

1. The parties locally compute $[\![\alpha]\!] = [\![x]\!] - [\![b]\!]$.
2. The parties run $\mathsf{open}([\![\alpha]\!])$ to obtain α.
3. Each party locally computes $[\![x^2]\!] = \alpha \cdot ([\![x]\!] + [\![b]\!]) + [\![b^2]\!]$.

Verification of a square pair using another. Similarly, one can use a random square $([\![b]\!], [\![b^2]\!])$ to verify the correctness of a given square $([\![x]\!], [\![x^2]\!])$ by working as follows:

1. The parties generate a random $\epsilon \in \mathbb{Z}_q \setminus \{0\}$.
2. The parties locally compute $[\![\alpha]\!] = [\![x]\!] - \epsilon[\![b]\!]$.
3. The parties run $\mathsf{open}([\![\alpha]\!])$ to obtain α.
4. Each party locally computes $[\![v]\!] = [\![x^2]\!] - \alpha \cdot ([\![x]\!] + \epsilon[\![b]\!]) - \epsilon^2[\![b^2]\!]$.
5. The parties run $\mathsf{open}([\![v]\!])$ to obtain v and accepts iff $v = 0$

We note that the above procedure works even if the random square is incorrect. Specifically, in this case the parties will accept with probability $\frac{2}{|\mathbb{F}|-1}$. See more details and proof in Appendix A.

2 The ZK Proof

Denote by $\mathcal{P}$ the prover and by $\mathcal{V}$ the verifier. In the protocol, both parties hold $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\vec{t} \in \mathbb{Z}_q^n$ as public input and the prover $\mathcal{P}$ holds $\vec{s} \in \mathbb{Z}_q^m$ as its private input. The aim of the prover $\mathcal{P}$ is to prove that $C(\vec{s}) = 1$ without leaking anything about $\vec{s}$.

Commitment scheme. To commit to a value x, the committing party chooses a random $r \in \{0,1\}^{128}$ and computes $c = \mathtt{com}(x, r) = HASH(x||r)$ and sends it to the other party or parties.

To open a commitment, the committing party sends r, x to the receiving parties which check that $c = HASH(x||r)$, outputting x if the equality holds and $\perp$ otherwise.

Pseudo random generation. pseudo-random numbers are generated in our protocol by choosing a random AES key and then use AES in counter-mode. Thus, in the following, when we say "seed is used to generate", we mean that seed is used as a key to AES.

2.1 A protocol based on Cut-and-Choose

We are now ready to describe our interactive protocol. Let H be a collision-resistant hash function.

- **Public input**: Both parties hold a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a vector $\vec{t} \in \mathbb{Z}_q^n$. In addition, the parties hold the parameters M, τ.
- **Private input**: The prover $\mathcal{P}$ holds a vector $\vec{s} \in \mathbb{Z}_q^m$.
- **The protocol**:
  - **Round 1:**
    1. For each $e = 1, \ldots, M$:
       (a) $\mathcal{P}$ chooses a master seed seed_e and use it to generate $\mathsf{seed}_{e,1}, \ldots, \mathsf{seed}_{e,N}$ by constructing a binary tree tree_e, with seed_e being its root and $\mathsf{seed}_{e,1}, \ldots, \mathsf{seed}_{e,N}$ being its leaves, where the seed on each node is used to generate the seeds of its two children.
       (b) For each $i \in \{1, \ldots, N-1\}$, $\mathcal{P}$ uses $\mathsf{seed}_{e,i}$ to generate $r_{e,i} \in \{0,1\}^{128}$ and the shares $b_{e,1,i}, \ldots, b_{e,m,i}, b^2_{e,1,i} \ldots, b^2_{e,m,i} \in \mathbb{Z}_q$.
       (c) $\mathcal{P}$ uses $\mathsf{seed}_{e,N}$ to generate $r_{e,N} \in \{0,1\}^{128}$ and $b_{e,1,N}, \ldots, b_{e,m,N} \in \mathbb{Z}_q$.
       (d) $\mathcal{P}$ computes $b_{e,k} = \sum_{i=1}^{N} b_{e,k,i}$ and define $b^2_{e,k,N} = (b_{e,k})^2 - \sum_{i=1}^{N-1} b^2_{e,k,i}$ for each $k \in \{1, \ldots, m\}$.
       (e) Let $\mathsf{state}_{e,i} = \mathsf{seed}_{e,i}$ for each $i \in \{1, \ldots, N-1\}$ and $\mathsf{state}_{e,N} = \mathsf{seed}_{e,N} || b^2_{e,1,N} || \cdots || b^2_{e,m,N}$. Then, for each $i \in \{1, \ldots, N\}$, $\mathcal{P}$ computes $\Gamma_{e,i} = \mathtt{com}(\mathsf{state}_{e,i}, r_{e,i})$.
       (f) Finally, $\mathcal{P}$ computes $h_e = H(\Gamma_{e,1} || \cdots || \Gamma_{e,N})$.
    2. $\mathcal{P}$ computes $h_\Gamma = H(h_1 || \cdots || h_M)$ and sends it to $\mathcal{V}$.
  - **Round 2:** $\mathcal{V}$ chooses a random challenge $E \subset \{1, \ldots, M\}$ such that $|E| = \tau$ and sends it to $\mathcal{P}$.
  - **Round 3:**
    1. Let $\bar{E} = \{1, \ldots, M\} \setminus E$. First, $\mathcal{P}$ chooses $\mathsf{seed}_{\bar{E}}$.
    2. For each $e \in \bar{E}$:
       (a) $\mathcal{P}$ uses $\mathsf{seed}_{\bar{E}}$ to generate $g_e \in \{0,1\}^{128}$.
       (b) For each $i \in \{1, \ldots, N-1\}$, $\mathcal{P}$ uses $\mathsf{seed}_{e,i}$ to generate $s_{e,1,i}, \ldots, s_{e,m,i}$.
       (c) $\mathcal{P}$ uses $\mathsf{seed}_{e,N}$ to generate $g_{e,N} \in \{0,1\}^{128}$.
       (d) For each $k \in \{1, \ldots, m\}$, $\mathcal{P}$ sets $s_{e,k,N} = s_k - \sum_{i=1}^{N-1} s_{e,k,i}$.
       (e) $\mathcal{P}$ computes $\alpha_{e,k,i} = s_{e,k,i} - b_{e,k,i}$ for each $i \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, m\}$.
       (f) $\mathcal{P}$ computes $\Omega_{e,N} = \mathtt{com}(s_{e,1,N} || \cdots || s_{e,m,N}, g_{e,N})$.
       (g) $\mathcal{P}$ computes $\Pi_e = \mathtt{com}(\alpha_{e,1,1} || \cdots || \alpha_{e,1,N} || \cdots || \alpha_{e,m,1} || \cdots || \alpha_{e,m,N}, g_e)$.
       (h) $\mathcal{P}$ computes $\alpha_{e,k} = \sum_{i=1}^{N} \alpha_{e,k,i}$ for each $k \in \{1, \ldots, m\}$.
    3. $\mathcal{P}$ computes $h_\pi = H(\Pi_{e_1} || \cdots || \Pi_{e_{|\bar{E}|}})$.
    4. $\mathcal{P}$ sends the following to $\mathcal{V}$:

* $\{\mathsf{seed}_e\}_{e \in E}$
* $\{\Omega_{e,N}\}_{e \in \bar{E}}$
* h_π

– **Round 4:** $\mathcal{V}$ works as follows:
  1. For each $e \in E$:
     (a) $\mathcal{V}$ uses seed_e to compute $r_{e,i}$ and $\mathsf{state}_{e,i}$ for each $i \in \{1, \ldots, N\}$ as described in Round 1. Then, it computes $\Gamma_{e,i}$ for each $i \in \{1, \ldots, N\}$ and $h_e = H(\Gamma_{e,1}|| \cdots ||\Gamma_{e,N})$.
  2. $\mathcal{V}$ chooses $\mathsf{seed}_{chal} \in \{0,1\}^{128}$. Then, for each $e \in \bar{E}$ he derives random coefficients

$$\beta_{e,1}, \ldots, \beta_{e,n}, \gamma_{e,1}, \ldots, \gamma_{e,m} \in \mathbb{Z}_q$$

     from seed_{chal} and then sends seed_{chal} to $\mathcal{P}$.

– **Round 5:**
  1. For each $e \in \bar{E}$ derive $\beta_{e,1}, \ldots, \beta_{e,n}, \gamma_{e,1}, \ldots, \gamma_{e,m} \in \mathbb{Z}_q$ from seed_{chal} the same way as $\mathcal{V}$ did.
  2. For each $e \in \bar{E}$:
     (a) For each $i \in \{1, \ldots, N\}$, $\mathcal{P}$ computes

$$o_{e,i} = \sum_{\ell=1}^{n} \beta_{e,\ell} \cdot (\frac{t_\ell}{N} - \sum_{k=1}^{m} a_{\ell,k} s_{e,k,i}) + \sum_{k=1}^{m} \gamma_{e,k} \cdot (\alpha_{e,k} \cdot (s_{e,k,i} + b_{e,k,i}) + b_{e,k,i}^2 - s_{e,k,i}) \quad (1)$$

     (b) $\mathcal{P}$ uses $\mathsf{seed}_{\bar{E}}$ to generate w_e. Then, it computes

$$\Psi_e = \mathtt{com}(o_{e,1}|| \cdots ||o_{e,N}, w_e).$$

  3. $\mathcal{P}$ computes $h_\psi = H(\Psi_{e_1}|| \cdots ||\Psi_{e_{|\bar{E}|}})$ and sends it to $\mathcal{V}$.

– **Round 6:** For each $e \in \bar{E}$: $\mathcal{V}$ chooses a random $\bar{i}_e \in \{1, \ldots, N\}$ and sends it to $\mathcal{P}$.

– **Round 7:** For each $e \in \bar{E}$: Let $I_e = \{1, \ldots, N\} \setminus \{\bar{i}_e\}$.
  1. $\mathcal{P}$ initialize an empty string seed_{tree_e}. Then, $\mathcal{P}$ traverses over the tree_e from the root to the leaves in the following way: if $\mathsf{seed}_{e,\bar{i}}$ is a descendant of the current node, then proceed recursively to its two children. If not, then add the seed of the current node to seed_{tree_e} and don't proceed to its children. If the current node is a leaf, then add its seed to seed_{tree_e}.
  2. $\mathcal{P}$ sends the following to $\mathcal{V}$:
     * $\mathsf{seed}_{\bar{E}}$
     * seed_{tree_e}
     * $\{\Gamma_{e,\bar{i}_e}\}_{e \in \bar{E}}$
     * $\{\{\alpha_{e,k,\bar{i}_e}\}_{k=1}^m\}_{e \in \bar{E}}$
     * $\{o_{e,\bar{i}_e}\}_{e \in \bar{E}}$
     * For each $e \in \bar{E}$: if $N \in I_e$ then send also $\{b_{e,k,N}^2\}_{k=1}^m$ and $\{s_{e,k,N}\}_{k=1}^m$

– **Round 8:** $\mathcal{V}$ does the following
  1. For each $e \in \bar{E}$:

(a) $\mathcal{V}$ computes $\{\mathsf{seed}_{e,i}\}_{i\in I_e}$ from seed_{tree_e}.
(b) $\mathcal{V}$ computes $r_{e,i}$, $\{b_{e,k,i}\}_{k=1}^m$ and $\{b_{e,k,i}^2\}_{k=1}^m$ for each $i \in I_e$ from $\mathsf{seed}_{e,i}$ (except for $\{b_{e,k,N}^2\}_{k=1}^m$ which was given to him explicitly).
(c) $\mathcal{V}$ computes $\mathsf{state}_{e,i}$ for each $i \in I_e$. Then, it computes $\Gamma_{e,i}$ for each $i \in I_e$.
(d) $\mathcal{V}$ computes $h_e = H(\Gamma_{e,1}||\cdots||\Gamma_{e,N})$ (recall that $\Gamma_{e,\bar{i}_e}$ was received from $\mathcal{P}$).
(e) If $N \in I_e$: $\mathcal{V}$ uses $\mathsf{seed}_{e,N}$ to generate $g_{e,N}$ and checks that $\Omega_{e,N} = \mathtt{com}(s_{e,1,N}||\cdots||s_{e,m,N}, g_{e,N})$. If not, it outputs reject and halts.
(f) For each $i \in I_e \setminus \{N\}$, $\mathcal{V}$ uses $\mathsf{seed}_{e,i}$ to generate $\{s_{e,k,i}\}_{k=1}^m$.
(g) For each $i \in I_e$ and $k \in \{1,\ldots,m\}$: $\mathcal{V}$ computes $\alpha_{e,k,i} = s_{e,k,i} - b_{e,k,i}$.
(h) $\mathcal{V}$ generates g_e from $\mathsf{seed}_{\bar{E}}$.
Then, it computes $\Pi_e = \mathtt{com}(\alpha_{e,1,1}||\cdots||\alpha_{e,1,N}||\cdots||\alpha_{e,m,1}||\cdots||\alpha_{e,m,N}, g_e)$.
(i) $\mathcal{V}$ computes $\alpha_{e,k} = \sum_{i=1}^N \alpha_{e,k,i}$ for each $k \in \{1,\ldots,m\}$.
(j) For each $i \in I_e$, $\mathcal{V}$ computes $o_{e,i}$ as in Eq. (1).
(k) $\mathcal{V}$ uses $\mathsf{seed}_{\bar{E}}$ to generate w_e. Then, it computes

$$\Psi_e = \mathtt{com}(o_{e,1}||\cdots||o_{e,N}, w_e)$$

(recall that $o_{e,\bar{i}_e}$ was received from $\mathcal{P}$).
(l) $\mathcal{V}$ checks that $\sum_{i=1}^N o_{e,i} = 0$. If not, it outputs reject and halts.

2. $\mathcal{V}$ computes $H(h_1||\cdots||h_M)$ and checks that it is equal to h_Γ. If not, it outputs reject and halts.
3. $\mathcal{V}$ computes $H(\Pi_{e_1}||\cdots||\Pi_{e_{|\bar{E}|}})$ and checks that it equals h_π received from the prover. if not, it outputs reject and halts.
4. $\mathcal{V}$ computes $H(\Psi_{e_1}||\cdots||\Psi_{e_{|\bar{E}|}})$ and checks that it equals h_ψ received from the prover. if not, it outputs reject and halts.
5. If $\mathcal{V}$ did not output reject till this step, then it outputs accept.

Soundness. We compute the probability that $\mathcal{V}$ outputs accept when $C(s) \neq 1$. Let c_1 be the number of off-line incorrect squares generated by $\mathcal{P}$. Let c_2 be the number of on-line emulations where $\mathcal{P}$ cheats. Thus, the success cheating probability is

$$\frac{\binom{M-c_1}{\tau}}{\binom{M}{\tau} \cdot N^{c_2} \cdot q^{M-\tau-c_1-c_2}}$$

Observe that if $q > N$, then the above is bounded by

$$\frac{\binom{M-c_1}{\tau}}{\binom{M}{\tau} \cdot N^{M-\tau-c_1}}$$

To ease readability we replace $M - c_1$ in the above equation with x, so we can rewrite the bound as

$$\frac{\binom{x}{\tau}}{\binom{M}{\tau} N^{x-\tau}}.$$

It now must hold that $\tau \leq x \leq M$: if the adversary chooses $c_1 = 0$ then this translates to $x = M$, while setting $c_1 > M - \tau$ will always lead to an abort.

We write out the binomial coefficents as factorials, which yields

$$\frac{x!(M-\tau)!}{(x-\tau)!M!N^{x-\tau}}$$

The fact that $x \leq M$ means that we can rewrite the above as

$$\frac{(M-\tau)\cdots(x-\tau+1)}{M\cdots(x+1)\cdot N^{x-\tau}} = \prod_{i=0}^{M-x-1} \frac{M-\tau-i}{M-i} \cdot \frac{1}{N^{x-\tau}} \leq \left(\frac{\sum_{i=0}^{M-x-1} \frac{M-\tau-i}{M-i}}{M-x}\right)^{M-x} \cdot \frac{1}{N^{x-\tau}}$$

by the inequality of quadratic and geometric means. Now we assume that $M = \delta\tau$ for some $\delta > 1$. Then every term in the sum can be upper-bounded by $\frac{\delta-1}{\delta}$ and we obtain

$$\left(\frac{\delta-1}{\delta}\right)^{c_1} \cdot \left(\frac{1}{N}\right)^{M-\tau-c_1}$$

Now as long as $\frac{\delta-1}{\delta} < 1/N$ the first derivative is never 0, i.e. it's monotone growing. But that means that the adversary should always choose c_1 as large as possible, which gives us a cheating probability of $\left(\frac{\delta-1}{\delta}\right)^{M-\tau}$.

Cost analysis. Denote the size in bits of the output of the hash function by $|hash|$, the size of a seed by $|\mathsf{seed}|$ and by $|\mathtt{com}|$ the size of the output of the commitment scheme.

The communication cost of messages sent from $\mathcal{P}$ to $\mathcal{V}$ is:

$$\begin{aligned} & |hash| + \tau \cdot |\mathsf{seed}| + (M-\tau)|\mathtt{com}| + 2|hash| + |\mathsf{seed}| \\ & \qquad + (M-\tau)(\log N|\mathsf{seed}| + |\mathtt{com}| + \log q \cdot m + \log q) = \\ = \quad & |hash| \cdot 3 + |\mathsf{seed}| \cdot (\tau + 1 + (M-\tau)\log N) + |\mathtt{com}| \cdot 2(M-\tau) + \log q \cdot (M-\tau)(m+1) \end{aligned}$$

which is also equivalent to

$$|hash| \cdot 3 + |\mathsf{seed}| + \tau \cdot |\mathsf{seed}| + (M-\tau) \cdot (2|\mathtt{com}| + \log q(m+1)) + (M-\tau)\log N \cdot |\mathsf{seed}|$$

concrete parameters. q, n, N, M, τ, m...

Ariel: TBD

2.2 A Protocol Based on Sacrificing

And here comes the second protocol. We here skip cut-and-choose and instead use the good old triple sacrifice approach.

- **Input**: Same as before
- **The protocol**:
  - **Round 1:**

1. For each $e = 1, \ldots, M$:
   (a) $\mathcal{P}$ chooses a master seed seed_e and use it to generate $\mathsf{seed}_{e,1}, \ldots, \mathsf{seed}_{e,N}$ by constructing a binary tree tree_e, with seed_e being its root and $\mathsf{seed}_{e,1}, \ldots, \mathsf{seed}_{e,N}$ being its leaves, where the seed on each node is used to generate the seeds of its two children.
   (b) For each $i \in \{1, \ldots, N-1\}$, $\mathcal{P}$ uses $\mathsf{seed}_{e,i}$ to generate $r_{e,i} \in \{0,1\}^{128}$ and the shares $s_{e,1,i}, \ldots, s_{e,m,i}, b_{e,1,i}, \ldots, b_{e,m,i}, b^2_{e,1,i} \cdots, b^2_{e,m,i} \in \mathbb{Z}_q$.
   (c) $\mathcal{P}$ uses $\mathsf{seed}_{e,N}$ to generate $r_{e,N} \in \{0,1\}^{128}$ and $b_{e,1,N}, \ldots, b_{e,m,N} \in \mathbb{Z}_q$.
   (d) For each $k \in \{1, \ldots, m\}$, $\mathcal{P}$ computes $b_{e,k} = \sum_{i=1}^{N} b_{e,k,i}$ and define $b^2_{e,k,N} = (b_{e,k})^2 - \sum_{i=1}^{N-1} b^2_{e,k,i}$.
   (e) For each $k \in \{1, \ldots, m\}$, $\mathcal{P}$ sets $s_{e,k,N} = s_k - \sum_{i=1}^{N-1} s_{e,k,i}$.
   (f) Let $\mathsf{state}_{e,i} = \mathsf{seed}_{e,i}$ for each $i \in \{1, \ldots, N-1\}$ and

   $$\mathsf{state}_{e,N} = \mathsf{seed}_{e,N} || s_{e,1,N} || \cdots || s_{e,m,N} || b^2_{e,1,N} || \cdots || b^2_{e,m,N}.$$

   Then, for each $i \in \{1, \ldots, N\}$, $\mathcal{P}$ computes $\Gamma_{e,i} = \mathtt{com}(\mathsf{state}_{e,i}, r_{e,i})$.
   (g) Finally, $\mathcal{P}$ computes $h_e = H(\Gamma_{e,1} || \cdots || \Gamma_{e,N})$.
2. $\mathcal{P}$ computes $h_\Gamma = H(h_1 || \cdots || h_M)$ and sends it to $\mathcal{V}$.

– **Round 2:** $\mathcal{V}$ chooses $\mathsf{seed}_{chal} \in \{0,1\}^{128}$. Then, for each $e \in \{1, \ldots, M\}$ he derives random coefficients

$$\epsilon_{e,1}, \ldots, \epsilon_{e,m}, \beta_{e,1}, \ldots, \beta_{e,n}, \gamma_{e,1}, \ldots, \gamma_{e,m} \in \mathbb{Z}_q$$

from seed_{chal} and then sends seed_{chal} to $\mathcal{P}$.

– **Round 3:**

1. For each $e \in \{1, \ldots, M\}$ derive $\epsilon_{e,1}, \ldots, \epsilon_{e,m}, \beta_{e,1}, \ldots, \beta_{e,n}, \gamma_{e,1}, \ldots, \gamma_{e,m} \in \mathbb{Z}_q$ from seed_{chal} the same way as $\mathcal{V}$ did.
2. $\mathcal{P}$ chooses random seed_{global}.
3. For each $e \in \{1, \ldots, M\}$:
   (a) $\mathcal{P}$ computes $\alpha_{e,k,i} = s_{e,k,i} - \epsilon_{e,k} \cdot b_{e,k,i}$ for each $i \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, m\}$.
   (b) $\mathcal{P}$ computes $\alpha_{e,k} = \sum_{i=1}^{N} \alpha_{e,k,i}$ for each $k \in \{1, \ldots, m\}$.
   (c) $\mathcal{P}$ uses seed_{global} to generate $g_e \in \{0,1\}^{128}$. Then, it computes

   $$\Pi_e = \mathtt{com}(\alpha_{e,1,1} || \cdots || \alpha_{e,1,N} || \cdots || \alpha_{e,m,1} || \cdots || \alpha_{e,m,N}, g_e)$$

   (d) For each $i \in \{1, \ldots, N\}$, $\mathcal{P}$ computes

   $$o_{e,i} = \sum_{\ell=1}^{n} \beta_{e,\ell} \cdot (t_\ell / N - \sum_{k=1}^{m} a_{\ell,k} s_{e,k,i}) \tag{2}$$

   (e) $\mathcal{P}$ uses seed_{global} to generate w_e. Then, it computes $\Psi_e = \mathtt{com}(o_{e,1} || \cdots || o_{e,N}, w_e)$.
   (f) For each $i \in \{1, \ldots, N\}$, $\mathcal{P}$ computes

   $$v_{e,i} = \sum_{k=1}^{m} \gamma_{e,k} \cdot \left(s_{e,k,i} - \alpha_{e,k} \cdot (s_{e,k,i} + \epsilon_{e,k} \cdot b_{e,k,i}) - (\epsilon_{e,k})^2 \cdot b^2_{e,k,i}\right) \tag{3}$$

(g) $\mathcal{P}$ uses seed_{global} to generate u_e. Then, it computes $\Theta_e = \mathtt{com}(v_{e,1}||\cdots||v_{e,N}, u_e)$.

4. $\mathcal{P}$ computes $h_\pi = H(\Pi_1||\cdots||\Pi_M)$.
5. $\mathcal{P}$ computes $h_\psi = H(\Psi_1||\cdots||\Psi_M)$.
6. $\mathcal{P}$ computes $h_\theta = H(\Theta_1||\cdots||\Theta_M)$
7. $\mathcal{P}$ sends the following to $\mathcal{V}$: h_π, h_ψ and h_θ.

– **Round 4:** For each $e \in \{1,\ldots,M\}$: $\mathcal{V}$ chooses a random $\bar{i}_e \in \{1,\ldots,N\}$ and sends it to $\mathcal{P}$.

– **Round 5:** For each $e \in \{1,\ldots,M\}$: Let $I_e = \{1,\ldots,N\} \setminus \{\bar{i}_e\}$.

1. $\mathcal{P}$ initialize an empty string seed_{tree_e}. Then, $\mathcal{P}$ traverses over the tree_e from the root to the leaves in the following way: if $\mathsf{seed}_{e,\bar{i}}$ is a descendant of the current node, then proceed recursively to its two children. If not, then add the seed of the current node to seed_{tree_e} and don't proceed to its children. If the current node is a leaf, then add its seed to seed_{tree_e}.
2. $\mathcal{P}$ sends the following to $\mathcal{V}$:
   * seed_{global}
   * seed_{tree_e}
   * $\{\Gamma_{e,\bar{i}_e}\}_{e\in\{1,\ldots,M\}}$
   * $\{\{\alpha_{e,k,\bar{i}_e}\}_{k=1}^m\}_{e\in\{1,\ldots,M\}}$
   * $\{o_{e,\bar{i}_e}\}_{e\in\{1,\ldots,M\}}$
   * $\{v_{e,\bar{i}_e}\}_{e\in\{1,\ldots,M\}}$
   * For each $e \in \{1,\ldots,M\}$: if $N \in I_e$ then send $\{b^2_{e,k,N}\}_{k=1}^m$ and $\{s_{e,k,N}\}_{k=1}^m$

– **Round 6:** $\mathcal{V}$ does the following

1. For each $e \in \{1,\ldots,M\}$:
   (a) $\mathcal{V}$ computes $\{\mathsf{seed}_{e,i}\}_{i\in I_e}$ from seed_{tree_e}.
   (b) $\mathcal{V}$ computes $r_{e,i}$, $\{s_{e,k,i}\}_{k=1}^m$, $\{b_{e,k,i}\}_{k=1}^m$ and $\{b^2_{e,k,i}\}_{k=1}^m$ for each $i \in I_e$ from $\mathsf{seed}_{e,i}$ (except for $\{s_{e,k,N}\}_{k=1}^m$ and $\{b^2_{e,k,N}\}_{k=1}^m$ which was given to him explicitly).
   (c) $\mathcal{V}$ computes $\mathsf{state}_{e,i}$ for each $i \in I_e$. Then, it computes $\Gamma_{e,i}$ for each $i \in I_e$.
   (d) $\mathcal{V}$ computes $h_e = H(\Gamma_{e,1}||\cdots||\Gamma_{e,N})$ (recall that $\Gamma_{e,\bar{i}_e}$ was received from $\mathcal{P}$).
   (e) For each $i \in I_e$ and $k \in \{1,\ldots,m\}$: $\mathcal{V}$ computes $\alpha_{e,k,i} = s_{e,k,i} - \epsilon_{e,k} \cdot b_{e,k,i}$.
   (f) $\mathcal{V}$ computes $\alpha_{e,k} = \sum_{i=1}^N \alpha_{e,k,i}$ for each $k \in \{1,\ldots,m\}$.
   (g) $\mathcal{V}$ generates g_e from seed_{global}.
   Then, it computes $\Pi_e = \mathtt{com}(\alpha_{e,1,1}||\cdots||\alpha_{e,1,N}||\cdots||\alpha_{e,m,1}||\cdots||\alpha_{e,m,N}, g_e)$.
   (h) For each $i \in I_e$, $\mathcal{V}$ computes $o_{e,i}$ as in Eq. (2).
   (i) $\mathcal{V}$ uses seed_{global} to generate w_e. Then, it computes $\Psi_e = \mathtt{com}(o_{e,1}||\cdots||o_{e,N}, w_e)$ (recall that $o_{e,\bar{i}_e}$ was received from $\mathcal{P}$).
   (j) $\mathcal{V}$ checks that $\sum_{i=1}^N o_{e,i} = 0$. If not, it outputs reject and halts.
   (k) For each $i \in I_e$, $\mathcal{V}$ computes $v_{e,i}$ as in Eq. (3).
   (l) $\mathcal{V}$ uses seed_{global} to generate u_e. Then, it computes $\Theta_e = \mathtt{com}(v_{e,1}||\cdots||v_{e,N}, u_e)$ (recall that $v_{e,\bar{i}_e}$ was received from $\mathcal{P}$).
   (m) $\mathcal{V}$ checks that $\sum_{i=1}^N v_{e,i} = 0$. If not, it outputs reject and halts.

2. $\mathcal{V}$ computes $H(h_1||\cdots||h_M)$ and checks that it is equal to h_Γ. If not, it outputs reject and halts.
3. $\mathcal{V}$ computes $H(\Pi_1||\cdots||\Pi_M)$ and checks that it is equal to h_π received from the prover. If not, it outputs reject and halts.
4. $\mathcal{V}$ computes $H(\Psi_1||\cdots||\Psi_M)$ and checks that it is equal to h_ψ received from the prover. If not, it outputs reject and halts.
5. $\mathcal{V}$ computes $H(\Theta_1||\cdots||\Theta_M)$ and checks that it is equal to h_θ received from the prover. If not, it outputs reject and halts.
6. If $\mathcal{V}$ did not output reject till this step, then it outputs accept.

Verifier-Optimized protocol. If we are willing to pay a bit in the proof size, we can let the prover send all the output shares $o_{e,i}$:

We can do the linear test for all parties of a simulated MPC instance together. For this, observe that Equation 1 is linear in all the values that depend on i. In the protocol we let the prover send $o_{e,1}, \ldots, o_{e,N}$ and the verifier checks that $0 = \sum_i o_{e,i}$. Instead of verifying each $o_{e,i}$ individually, we want to do establish this fact with lower overhead (as computing each $o_{e,i}$ involves a huge matrix multiplication). For this, we use the following Lemma:

Lemma 1. *Let $a_1, \ldots, a_{m+1} \in \mathbb{Z}_q$ and $f : \mathbb{Z}_q^m \to \mathbb{Z}_q, (\tau_1, \ldots, \tau_m) \mapsto a_{m+1} + \sum_{i=1}^m a_i\tau_i$. Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

1. *$\mathcal{A}$ picks a function f according to the above definition, $\{\tau_{1,j}, \ldots, \tau_{m,j}\}_{j\in[N]}$ as well as $c_1, \ldots, c_N \in \mathbb{Z}_q$ and sends these to the challenger.*

2. *The challenger picks $\alpha_1, \ldots, \alpha_N \overset{\$}{\leftarrow} \mathbb{Z}_q$ and tests that*

$$\sum_{j\in[N]} \alpha_j \cdot c_j = (\sum_{j\in[N]} \alpha_j) \cdot a_{m+1} + \sum_{i\in[m]} a_i \cdot \sum_{j\in[N]} \alpha_j \cdot \tau_{i,j}$$

If so then he outputs 1, otherwise 0.

3. *The adversary wins if $\mathcal{C}$ outputs 1 and $\exists j \in [N]:\ c_j \neq f(\tau_{1,j}, \ldots, \tau_{m,j})$.*

Then the adversary wins with probability $1/q$.

Proof. Assume that $\mathcal{A}$ wins. We alternatively write $c_j = f(\tau_{1,j}, \ldots, \tau_{m,j}) + \Delta_j$, where Δ_j is non-zero for at least one $j \in [N]$. We can rewrite the checking equation as

$$\begin{aligned}
\sum_{j\in[N]} \alpha_j \cdot c_j &= \sum_{j\in[N]} \alpha_j \cdot (f(\tau_{1,j}, \ldots, \tau_{N,j}) + \Delta_j) \\
&= \sum_{j\in[N]} \alpha_j \cdot f(\tau_{1,j}, \ldots, \tau_{N,j}) + \sum_{j\in[N]} \alpha_j \cdot \Delta_j \\
&= (-1 + \sum_{j\in[N]} \alpha_j) \cdot a_{m+1} + f(\sum_{j\in[N]} \alpha_j \cdot \tau_{1,j}, \ldots, \sum_{j\in[N]} \alpha_j \cdot \tau_{N,j}) + \sum_{j\in[N]} \alpha_j \cdot \Delta_j \\
&= (\sum_{j\in[N]} \alpha_j) \cdot a_{m+1} + \sum_{i\in[m]} a_i \cdot \sum_{j\in[N]} \alpha_j \cdot \tau_{i,j} + \sum_{j\in[N]} \alpha_j \cdot \Delta_j
\end{aligned}$$

and so $\mathcal{C}$ will output 0 unless $0 = \sum_{j\in[N]} \alpha_j \cdot \Delta_j$. But since the α_j are chosen uniformly at random and by assumption $\vec{\Delta} = (\Delta_1, \ldots, \Delta_N) \neq \vec{0}$, $\vec{\Delta}$ lies in the kernel of the induced linear map only with probability $1/q$. □

In our protocol, we check

$$o_{e,i} = \sum_{\ell=1}^{n} \beta_{e,\ell} \cdot (t_\ell/N - \sum_{k=1}^{m} a_{\ell,k} s_{e,k,i})$$

$N-1$ times, performing $N-1$ matrix multiplications. By the above Lemma, the verifier can instead sample $\phi_1, \ldots, \phi_{N-1} \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random and test that

$$\sum_{i=1}^{N-1} \phi_i \cdot o_{e,i} = (\sum_{i=1}^{N-1} \phi_i) \cdot \sum_{\ell=1}^{n} \beta_{e,\ell} \cdot t_\ell/N - \sum_{\ell=1}^{n}\sum_{k=1}^{m} \alpha_{\ell,k} \cdot (\sum_{i=1}^{N-1} \phi_i \cdot s_{e,k,i}).$$

Though the sums on the right side $(\sum_{i=1}^{N-1} \phi_i \cdot s_{e,k,i})$ must also be computed, this amounts to computing m inner products where one vector is fixed among all instances.

Soundness. Let c_1 be the number of executions where $\mathcal{P}$ cheats in computing the view of some party. Let c_2 be the number of executions where $\mathcal{P}$ cheats in the square computation. Then, the probability that $\mathcal{V}$ accepts is bounded by

$$p_2(N, M, q) = \max_{c_1,c_2} | = \frac{1}{N^{c_1}} \cdot \left(\frac{2}{q}\right)^{c_2} \cdot \frac{1}{q^{M-c_2-c_1}} = \frac{1}{N^{c_1}} \cdot \frac{2^{c_2}}{q^{M-c_1}}$$

We can assume that $0 \le c_1 + c_2 \le M$, because an adversary that cheats twice in an instance has a higher chance of being caught while achieving the same result. Therefore

$$p_2(N, M, q) \le \max_{c_1} \frac{1}{N^{c_1}} \cdot \left(\frac{2}{q}\right)^{M-c_1}$$

If $N \le q/2$ then it's always beneficial to simply set $c_1 = M$, otherwise $c_1 = 0$ is the best option. This seems to be always better than the first protocol, and the communication cost is always smaller due to lack of Cut and Choose

Cost analysis. Denote the size in bits of the output of the hash function by $|hash|$, the size of a seed by $|\mathsf{seed}|$ and by $|\mathtt{com}|$ the size of the output of the commitment scheme.

The communication cost of messages sent from $\mathcal{P}$ to $\mathcal{V}$ is:

$$|hash| + 3|hash| + |\mathsf{seed}| + M(|\mathsf{seed}|\log N + |\mathtt{com}| + m \cdot \log q + 2\log q) =$$
$$|hash| \cdot 4 + |\mathsf{seed}| \cdot (1 + M \log N) + |\mathtt{com}| \cdot M + \log q \cdot M(m+2)$$

which is equivalent to

$$hash| \cdot 4 + |\mathsf{seed}| + M \cdot (|\mathtt{com}| + \log q(m+2)) + M \log N \cdot |seed|$$

concrete parameters. $q, n, N, M, \tau, m, ...$
Ariel: TBD

A Verification of a Square Pair Using Another

Let us first prove that our approach works. Therefore, consider the following game:

Game 1. *Let $\mathcal{P}, \mathcal{V}$ be defined as above and* `com` *be a commitment scheme.*

1. *$\mathcal{P}$ chooses $\{a_i, b_i, \hat{a}_i, \hat{b}_i\}_{i\in[N]}$ from $\mathbb{F}$, computes $\Gamma_i \leftarrow \mathtt{com}(a_i, b_i, \hat{a}_i, \hat{b}_i)$ and sends $\{\Gamma_i\}_{i\in[N]}$ to $\mathcal{V}$.*
2. *$\mathcal{V}$ samples $x \xleftarrow{\$} \mathbb{F} \setminus \{0\}$ and sends it to $\mathcal{P}$.*
3. *$\mathcal{P}$ computes $\alpha_i \leftarrow a_i - x \cdot b_i, \beta_i \leftarrow \hat{a}_i - x^2\hat{b}_i - \alpha_i(a_i + xb_i)$ and sends $\{\alpha_i, \beta_i\}_{i\in[N]}$ and sends these to $\mathcal{V}$.*
4. *$\mathcal{V}$ samples $j \xleftarrow{\$} \{1, \ldots, N\}$ and sends it to $\mathcal{P}$.*
5. *$\mathcal{P}$ sets $\overline{E} = \{1, \ldots, N\} \setminus \{j\}$ and opens $\{\Gamma_i\}_{i\in\overline{E}}$ as $(a'_i, b'_i, \hat{a}'_i, \hat{b}'_i)$ to $\mathcal{V}$.*
6. *$\mathcal{V}$ outputs* accept *if the commitments open correctly, $\alpha_i = a'_i - xb'_i$, $\beta_i = \hat{a}'_i - x^2\hat{b}'_i - \alpha_i(a'_i + xb'_i)$ and $0 = \sum_i \beta_i$, and otherwise outputs* reject.
7. *$\mathcal{P}$ wins if $\sum_i \hat{a}_i \neq (\sum_i a_i)^2$ and $\mathcal{V}$ outputs* accept.

It is obvious that the above test leaks no information about $a = \sum_i a_i$ since each a_i is perfectly blinded by b_i in α_i. Carsten: The β_i might be a problem, but in the end we will release a random linear combination of these anyway. So we can add a random sharing of 0..

Lemma 2. *Assume that* `com` *is binding, then $\mathcal{P}$ wins the above game with probability at most $1/N + 2/(|\mathbb{F}| - 1)$.*

Proof. In the following, we always assume that `com` is binding, and therefore $a'_i = a_i, b'_i = b_i, \hat{a}'_i = \hat{a}_i$ and $\hat{b}'_i = \hat{b}_i$.

We write $\sum_i \hat{a}_i = (\sum_i a_i)^2 + \Delta_a$ and $\sum_i \hat{b}_i = (\sum_i b_i)^2 + \Delta_b$. Assume that $\mathcal{P}$ computes α_i, β_i correctly, but $\Delta_a \neq 0$ while $\mathcal{V}$ accepts. Let us define $a = \sum_i a_i, b = \sum_i b_i, \hat{a} = \sum_i \hat{a}_i, \hat{b} = \sum_i \hat{b}_i$. Then by step 6 we obtain that

$$\begin{aligned} 0 = \sum_i \beta_i &= \hat{a} - x^2\hat{b} - (a + xb) \cdot (a - xb) \\ &= \Delta_a - x^2\Delta_b \end{aligned}$$

and therefore $x^2 = \Delta_a/\Delta_b$. There exist exactly $\frac{|\mathbb{F}|-1}{2}$ squares in $\mathbb{F}$, so fixing Δ_a, Δ_b in advance means predicting x^2, which is chosen uniformly at random from all squares. Hence here $\mathcal{P}$ will only succeed with probability $2/(|\mathbb{F}| - 1)$.

On the other hand, the above only holds if $\mathcal{P}$ indeed computes α_i, β_i correctly. $\mathcal{P}$ may do so by sending incorrect α_i, β_i, but $\mathcal{V}$ will choose to open Γ_i with probability $\frac{N-1}{N}$. By a union bound, the aforementioned success probability follows. □

Carsten: Note: we do not have to send all α_i, β_i to $\mathcal{V}$ - it suffices to send a collision-resistant hash of all these values. $\mathcal{P}$ must only send α_j, β_j while all remaining values can be recomputed by $\mathcal{V}$.