# Efficient, UC-secure and Publicly Auditable MPC from OLE & VOLE-in-the-head

Carsten Baum[1] and Chiara-Marie Zok[1]

Technical University of Denmark, {cabau,chizo}@dtu.dk

**Abstract.** Secure Multiparty Computation (MPC) computes on private input data, but generally does not guarantee correctness of the output towards third parties. This property, also called public auditability, was first studied explicitly by Baum et al. (SCN 2014). Their work and its follow-ups generate a Non-Interactive Zero-Knowledge proof of correctness of the MPC outcome during the MPC protocol, ensuring validity of the output even if all parties are corrupted.

In this work, we revisit and improve the MPC with Public Auditability blueprint. While the original work uses a version of the SPDZ MPC protocol with expensive lattice-based preprocessing, our construction combines any generic OLE-based preprocessing with a publicly verifiable somewhat linearly homomorphic commitment scheme from VOLE-in-the-head in a non-trivial way. Our commitment scheme relies solely on random oracle calls instead of previously used linearly homomorphic commitments based on structured Public-Key assumptions.

## 1 Introduction

Secure multi-party computation (MPC) allows a set of mutually distrusting parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ to evaluate a function on their inputs in a privacy-preserving way. When the computation is completed, participants learn nothing except the output of the function. MPC protocols can be used in a variety of contexts such as private voting, auctions or privacy preserving machine learning [14,26,29,1,27] (to just mention some examples). A widely used paradigm to construct secret sharing-based MPC protocols divides computation into a preprocessing and online phase, with the most popular approaches following the BDOZ and SPDZ protocols [21,13,20]. There, during the computationally heavy preprocessing phase, correlated randomness is generated that is then consumed during the lightweight online phase.

In addition to privacy, MPC guarantees that the output of the computation is correct. This is especially important when MPC is used for computations with large impact such as e-voting. However, MPC guarantees hold only to the participating parties and only for corruptions up to the corruption threshold, which is usually strictly smaller than the total number of parties. Publicly Auditable MPC, first introduced in [8], solves this by giving an external party the possibility to audit the transcript after the computation took place. Their MPC protocol

(and follow-up works such as [32]) ensures privacy as long as at least one party is honest, and correctness even if all parties are corrupted. This is achieved in the presence of an external, incorruptible auditor that does not have a secret state and that is not involved in the computation.

Public verifiability can trivially be realized if each party constructs a Non-Interactive Zero-Knowledge proof (NIZK) of correctness while it participates in the MPC protocol. When instantiating this with generic NIZKs [2,12] this leads to suboptimal performance due to the approach being non-black box in the underlying MPC protocol or having to produce the NIZK inside MPC. [8] avoid this by relying on the [21] MPC protocol, that has an information-theoretic online phase where all protocol messages are openings of linear secret sharings. By letting all MPC messages be sent through a bulletin board, which works as a broadcast with memory, the protocol messages can be retained by anyone including the auditor.

[8] then lets each party commit to its shares using linearly homomorphic Pedersen commitments. This allows anyone to establish correctness of the reconstruction of secret sharings and thus the correctness of the MPC output. The construction relies on lattice-based threshold somewhat homomorphic encryption (SHE) for a publicly verifiable preprocessing. The linear homomorphism of the commitments is used to achieve verifiability of the online phase, but this also means that their protocol can only be instantiated for computations over fields as large as Discrete Logarithm (DLOG)-hard groups. In addition, they rely on SHE with computationally costly proofs of verifiable decryption where the plaintext space must be as large as the DLOG-hard group of the commitments.

### 1.1 Our Contributions

In this work, we improve the [8] blueprint in multiple ways. First, we show that the underlying MPC scheme can be instantiated using recent Pseudorandom Correlation Generator (PCG)-based preprocessing for Oblivious Linear Evaluation (OLE) correlations, reducing the communication overhead due to triple generation. Moreover, we show that the recent Vector-OLE-in-the-Head (VOLEitH) paradigm allows for the construction of suitable linearly homomorphic commitments that make the MPC scheme publicly verifiable without using any assumptions but a Random Oracle. This avoids both expensive verifiable decryption as well as commitments based on public-key assumptions. Finally, we prove our constructions secure in the Universal Composability (UC) framework.

*Better UC one-time homomorphic commitments.* It is well-known that UC-secure VOLE-based homomorphic commitments can easily be constructed based on a PCG preprocessing [35,10,34]. However, these are designated-verifier commitments that require a private state, making them unsuitable for Auditable MPC. We observe that VOLE-in-the-head ZK proofs implicitly define linearly homomorphic commitments, but where such commitments only can be opened

once. We formalize and analyze their construction in the UC framework and construct the first UC-secure one-time linearly homomorphic commitment scheme[1] from VOLEitH. These commitments can, in turn, be constructed only assuming a Random oracle, and we provide a UC formalization that permits their use in non-interactive protocols based on the Fiat-Shamir transform. Moreover, we show that our commitments can be opened more than once: basically, a sender can send openings *to the future* by committing to the opening using the Random Oracle and thus open the openings at a later point.

*Efficient Auditable MPC for any (large) field using OLE.* [8] relies on the linear homomorphism and public verifiability of Pedersen commitments, and also that those commitments can be opened repeatedly without weakening security. We simplify the preprocessing, redesign how the MPC protocol uses the linearly homomorphic commitments, and show how to reconcile the one-time property with the required verifiability of the openings. Namely, we observe that it is sufficient if the commitment scheme allows for *two* reliable openings: one to verify all opened secret sharings in the MPC protocol, and another to verify the output. The second opening can be realized from sending openings *to the future* which our commitment scheme permits. Using the linearity of the commitment scheme, we implicitly construct a computationally lightweight NIZK tailored to our MPC scheme that efficiently proves honest behavior of parties.

### 1.2 Technical Overview

*High-level structure of the MPC protocol.* We begin by describing the architecture of our MPC protocol from a high level, before showing how the individual parts are constructed. For a visual overview, see Figure 1. All our protocols make use of the global bulletin board $\mathcal{F}_{\mathsf{GBB}}$ for communication and almost all protocols use the global random oracle $\mathcal{F}_{\mathsf{GpRO}}$ for various purposes. The transcript on $\mathcal{F}_{\mathsf{GBB}}$ can be used by any auditor to verify the correctness of the computation as soon as the output is sent to the bulletin board. The MPC protocol follows the offline-online paradigm. The online phase, formalized by functionality $\mathcal{F}_{\mathsf{Online}}$ (see Section 6) is realized through the protocol $\Pi_{\mathsf{Online}}$.
For a fixed circuit $\mathcal{C}$ the parties first generate all required correlated randomness, and commitments to it, using the functionality $\mathcal{F}_{\mathsf{Offline}}$. This correlated randomness is consumed in the circuit computation. We realize $\mathcal{F}_{\mathsf{Offline}}$ through the protocol $\Pi_{\mathsf{Offline}}$ where we use the OLE functionality $\mathcal{F}_{\mathsf{OLE}}$ to generate Beaver Triples and a functionality $\mathcal{F}_{\mathsf{HOTCom}}$ to commit to random values and triples. To realize $\mathcal{F}_{\mathsf{HOTCom}}$, which formally describes our VOLEitH-based commitments with public verifiability, we first construct a protocol $\Pi^{1-1}_{\mathsf{HOTCommit}}$ that realizes a version of $\mathcal{F}_{\mathsf{HOTCom}}$ (called $\mathcal{F}^{i}_{\mathsf{HOTCom}}$). This single-sender single-receiver commitment is built on top of $\mathcal{F}_{\mathsf{WVOLE}}$, a functionality that generates *weak* VOLE

[1] One-time UC-secure linearly homomorphic commitments from random oracles have previously been constructed in [18], but our construction yields concretely better parameters due to the use of VOLEitH.

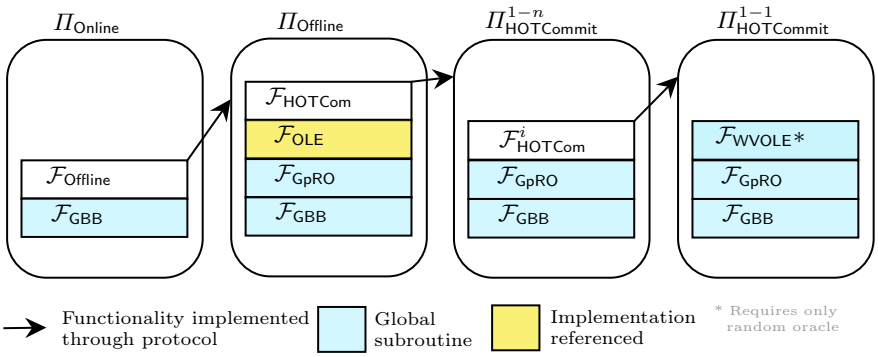


Fig. 1: Overview of protocols and functionalities.

correlations for a sender and a receiver. We then show how to use multiple $\mathcal{F}^{i}_{\mathsf{HOTCom}}$ instances to realize $\mathcal{F}_{\mathsf{HOTCom}}$ in the single-sender multi-receiver setting based on the protocol $\Pi^{1-n}_{\mathsf{HOTCommit}}$. This two-step construction of $\mathcal{F}_{\mathsf{HOTCom}}$ is necessary to allow senders to reliably open commitments to individual receivers. We finally realize $\mathcal{F}_{\mathsf{WVOLE}}$ by carefully formalizing and analyzing the previous VOLEitH construction in the UC framework. For $\mathcal{F}_{\mathsf{OLE}}$ we give references for existing implementations. Hence our construction only relies on a global random oracle, a bulletin board and OLE.

*OLE & MPC.* We use a version of the BDOZ MPC protocol [13] for the online phase. In BDOZ, each party $\mathcal{P}_i$ of the $n$ parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ holds a share $x_i \in \mathbb{F}$ of a secret $x$ such that $x = x_1 + \cdots + x_n$ where $\mathbb{F}$ is a finite field. To make sure parties do not alter their shares, BDOZ uses pair-wise linearly homomorphic MACs. This prevents parties from revealing inconsistent shares $x'_i \neq x_i$ of the secret, while it still allows all parties to perform linear computations on the shared secrets. BDOZ also uses preprocessed and shared multiplication triples together with Beaver's circuit randomization technique [11] to perform multiplications on secret-shared data. Recently, Rachuri & Scholl [30] have shown how to implement a preprocessing phase for SPDZ-type protocols using pair-wise Programmable PCGs that generate Oblivious Linear Evaluation correlations over a large field $\mathbb{F}$. We use their approach for preprocessing, but with heavy modifications to achieve public verifiability.

*VOLEs & linearly homomorphic MACs.* In the BDOZ MACs, the sender (the party holding the share) has the share $x_i \in \mathbb{F}$ and a MAC $w_i \in \mathbb{F}$, while the receiver (the party establishing correctness of the opening) holds $\Delta, v_i \in \mathbb{F}$ such that $w_i = v_i + x_i\Delta$. To reveal the share in a verifiable way, the sender sends $x_i, w_i$ to the receiver, who checks that the correlation holds. For all shares of $\mathcal{P}_i$ towards the same receiver $\mathcal{P}_j$, the receiver uses the same $\Delta$ in all correlations (making the MAC scheme linearly homomorphic). Put differently, correctly established MACs on secret shares form a Vector Oblivious Linear Evaluation (VOLE) cor-

relation. However, security of the MACs fails in the public verifiability setting as $\Delta$ becomes known to the sender if all parties are corrupted.

*Making VOLE MACs into publicly verifiable commitments.* Public verifiability for VOLE correlations was recently addressed in [7]. Their work extends the PCG-based ZK proofs of [35,10,34] towards public verifiability of VOLE-based NIZKs. [7] achieve this by observing that a weaker form of VOLE is sufficient for VOLE-based MACs used in NIZKs, viewing these MACs as commitments. The sender obtains $x_i, w_i$ already during the commitment phase, and can conduct arbitrary (at this point unverifiable) openings of commitments. Once all operations on commitments and their openings are completed, $\Delta$ is generated as a random oracle output. Finally, based on this choice of $\Delta$, the receiver learns $v_i$ which allows it to verify the openings.

In [7] the focus lies on achieving VOLE-based NIZK proofs. We formalize the properties of the commitment scheme that [7] actually achieves, and prove that their construction yields one-time linear UC-secure commitments. Towards this, we have to explicitly apply the Fiat-Shamir transform to construct the commitment scheme instead of directly compiling to NIZKs as in previous works. As the one-time property is insufficient for use in MPC, we use that, during an opening, the sender can commit to its opening $x_i, w_i$ using a regular UC commitment instead of sending it directly to the receiver. This gives the sender a limited way to make openings even after the verifier's key $\Delta$ has been publicly revealed. We formalize this property, which we call *delayed openings*, and use it in our construction.

Having only single-receiver commitments is insufficient for use in MPC protocols, where we want each party to be committed to its shares *consistently* towards all other parties. We therefore use a consistency check based on a universal hash, combined with a global bulletin board that we use as broadcast channel. To reliably avoid cheating even if all parties are corrupted, we determine the challenge used in the check using the Fiat-Shamir transform applied to all commitments previously made. In addition, we show how the check integrates with our commitment scheme with its constraints on verifiable openings.

*Commitments in MPC.* We then integrate our new commitment scheme into the MPC protocol. There, the parties will use OLEs to generate secret sharings of multiplication triples. We then let parties post commitments to their shares on a public bulletin board. This will also be used by parties to open commitments during evaluation of the MPC protocol, and we check correctness of the multiplication triples akin to [28]. All verification steps of the MPC scheme, as well as the actual computation, can be combined into a single verifiable opening. However, in order to achieve input independence (as in previous works), we only reveal the output of the computation once all consistency checks passed.

*Universal Composition and Fiat-Shamir.* We formalize lower-level primitives such as weak VOLE and one-time commitments in UC, give a composable con-

struction and analyze all security guarantees modularly. This clashes with the use of the Fiat-Shamir transform when generating challenges in many ways: for example, to generate a challenge that tests if multiplication triples are well-formed, the challenge must be computed based on the state of the UC functionality which holds the shares. However, normal UC functionalities do not offer such a mechanism, and a (global) random oracle cannot hash such a state either. While this could in theory be resolved by combining all primitives into a single functionality, this approach would strongly contradict the modularity of UC.

We therefore carefully model a *challenge generation* mechanism into our functionalities where parties (and the auditor) can obtain unpredictable challenges for higher-level protocols based on the functionality's state. However, any rushing adversary (or in particular any adversary if all parties are corrupted) can choose its inputs to the functionality, observe which challenge this generates, and adaptively change its choice of inputs. This mimics a real-world attack where an attacker simulates the impact of its messages onto random oracle outputs before posting messages to the bulletin board. Our UC modeling permits such attacks by allowing attackers to obtain this information or even reset functionalities.

*Communication complexity and Open problems.* In order to quantify the cost of our protocol, we have analyzed its communication complexity in terms of bits sent per party and total number of rounds and separated them by offline and online phase. The offline phase can be completed in 5 rounds while the online phase takes $4 + \mathcal{D}$ rounds, where $\mathcal{D}$ is the circuit depth. For details see Appendix C. While our work addresses some drawbacks of the state-of-the-art, other questions remain open. For example, our construction does not yet work for binary circuits. We describe limitations and open problems in Appendix A.

### 1.3 Related Work

Cunningham et al. [19] extended [8] by the use of so-called Commitment Enhanced Secret Sharing. The authors introduce the notion of completely identifiable auditability by adding linear homomorphic commitments to each additive share. [19] uses Pedersen commitments so the construction is not post-quantum secure. Schoenmakers and Veeningen [32] use a threshold version of Paillier and generic NIZK, making their work also not post-quantum secure. The authors achieve *universally verifiable* MPC, which extends public auditability to prove correctness of an encryption of a secret result. By relying on a generic NIZK, both the proof size and prover runtime cannot be optimized for the specific MPC scheme. In [31], the authors present an enhanced version of [8,19] that uses lattice-based commitments and additionally supports lower corruption thresholds towards robustness. Their work heavily relies on NIZKs during the offline phase to show correctness of lattice-based commitments, ciphertexts and multiplications and requiring a CRS for the commitment scheme. In comparison, our offline phase is essentially equivalent to the OLE-based preprocessing of [30] and works using any UC-secure OLE. Parties simply commit to the outcomes and run a very lightweight and standard multiplication triple check.

Baldimtsi et al. [4] define the notion of crowd-verifiable zero-knowledge (CVZR) proofs to achieve what they call end-to-end verifiable MPC such that cheating can be detected even if the setup assumption of a well-formed CRS does not hold. Similar to [8], [32] and [19] Baldimtsi et al. consider separation of input and computation parties but in contrast to prior work they also consider corruption of input parties. Our construction does not support separation of input and computation parties. They prove their construction secure in the random oracle model without using trapdoor commitments. They state that their construction as is, is secure when at least one computation party is honest but can be easily altered to achieve public audibility following the notion of [8] when the computation result is publicly decrypted. However, they show that their approach is limited to only certain securely computed functions, and they require super-polynomial security assumptions. In our work, we boost single-receiver commitments to multi-receiver commitments. To combat cheating adversaries that provide inconsistent shares [22] also used pairwise homomorphic commitments. Their construction is secure in the dishonest majority setting and can be instantiated using minimal assumptions. In summary, all previous works either are not post-quantum secure, only support a subset of all possible functions or do not support full corruption. Our construction instead only relies on random oracles, works for all circuits and is secure even if all parties are corrupted. We provide a comparison in Table 1.

| | Ours | [8] | [4] | [19] | [22] | [31] | [30] | [32] |
|---|---|---|---|---|---|---|---|---|
| Auditable | ✓ | ✓ | (✓) | ✓ | X | ✓ | X | ✓ |
| Composable | UC | UC | UC | UC | UC | UC | UC | X |
| Assumptions (except RO) | OLE | Lattice+ DLOG | subexp. TPKE | Lattice+ DLOG | PRG+ OT | Lattice | pPCGs | Paillier |
| Setup | X | CRS | X | CRS | X | CRS | X | TPKE |
| PQ | ✓ | X | X | X | ✓ | ✓ | ✓ | X |

Table 1: Comparison of related work. All constructions that do not support full corruption are secure in the dishonest majority setting. TPKE = Threshold Public Key Encryption, pPCG = programmable PCG.

## 2 Preliminaries and Notation

We use $[x..y]$ to denote the interval $\{x, x+1, \ldots, y\}$ and write $[x]$ for $[1..x]$. We write vectors in lower-case bold letters such as $\mathbf{x}$, and matrices as upper-case bold letters such as $\mathbf{A}$. For a matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$ we use $\mathbf{A}_i$ to denote the $i$'th row and $\mathbf{A}^j$ for the $j$th column. $\mathbf{A}[i, j]$ denotes the element at row $i$ and column $j$ of the matrix $\mathbf{A}$. Similarly, we use $\mathbf{x}[i]$ or $x_i$ to denote the $i$th element of $\mathbf{x}$.

We use $\lambda$ for the security parameter. Parties in this work are usually denoted using the symbol $\mathcal{P}$. We write $v^i$ to indicate that $v$ is either sent by or sent to $\mathcal{P}_i$. For a value $v$ with index $id$ we write $v_{id}$ and with $v^i_{id}$ we denote a value $v$ with index $id$ that was sent either to or by $\mathcal{P}_i$. In our protocols and functionalities,

we use lists (using notation `li`) and maps (using notation `mp`) to store values. If elements have to be addressable by an ID they are stored inside maps and otherwise in lists. Flags (which are essentially publicly known variables with notation `fl`) are used to indicate states of functionalities and protocols to ensure that certain operations can only be performed when preliminary actions are completed. Additional preliminaries can be found in Appendix B.

### 2.1 Linear Codes

Let $\mathbb{F}$ be a finite field and $n_{\mathcal{C}}, k_{\mathcal{C}} \in \mathbb{N}$ be integers. A *linear code* $\mathcal{C} \subset \mathbb{F}^{nc}$ is a $k_{\mathcal{C}}$-dimensional vector subspace of $\mathbb{F}^{nc}$, $1 \leq k_{\mathcal{C}} \leq n_{\mathcal{C}}$. Defining

$$d_{\mathcal{C}} = \min\{\#\text{non-zero indices of } \mathbf{c} \mid \mathbf{c} \in \mathcal{C} \setminus \{\mathbf{0}\}\},$$

we say that $\mathcal{C}$ is a linear code over $\mathbb{F}$ with length $n_{\mathcal{C}}$, dimension $k_{\mathcal{C}}$ and (minimum) distance $d_{\mathcal{C}}$ or $[n_{\mathcal{C}}, d_{\mathcal{C}}, k_{\mathcal{C}}]$-code for short.

A matrix $\mathbf{G} \in \mathbb{F}^{k_{\mathcal{C}} \times n_{\mathcal{C}}}$ is a *generator matrix* of $\mathcal{C}$ if $\mathbf{x}^T\mathbf{G}$ generates $\mathcal{C}$, i.e. $\mathcal{C} = \{\mathbf{x}^T\mathbf{G} \mid \mathbf{x} \in \mathbb{F}^{k_{\mathcal{C}}}\}$. If $\mathbf{G}$ is of the form $[\mathbf{I}_{k_{\mathcal{C}}}|\mathbf{A}]$, where $\mathbf{I}_{k_{\mathcal{C}}}$ is the $k_{\mathcal{C}} \times k_{\mathcal{C}}$ identity matrix, then $\mathbf{G}$ is called *systematic*.

### 2.2 UC & Functionalities

We model protocols and their security in this work using the Universal Composability (UC) framework [16]. See Appendix B.2 for a short UC introduction.

*Notation.* Throughout this work, we will denote functionalities using $\mathcal{F}$, protocols using $\Pi$, the adversary as $\mathcal{A}$ and simulators as $\mathcal{S}$. The operations of a functionality act as abstract interfaces that can be invoked by parties and the adversary, independent of any protocol implementation. Those interfaces are usually denoted in **Bold Text** inside a functionality. A first interaction with a functionality $\mathcal{F}$ for such a interface can always either be triggered by an honest party, or by the adversary $\mathcal{A}$ playing the role of a dishonest party. Always stating this fact, however, would make our functionalities harder to read. To simplify notation, we write that a message can be sent by a possibly corrupted party, meaning that formally $\mathcal{A}$ sends the message on behalf of the corrupted party.

*Functionalities.* We use ideal functionalities for random oracles $\mathcal{F}_{\mathsf{GpRO}}$ and public bulletin boards $\mathcal{F}_{\mathsf{GBB}}$, which can be found in Figure 3 and Figure 2 respectively. Both are modeled as global subroutines. Note that $\mathcal{F}_{\mathsf{GBB}}$ as a bulletin board allows external parties to read from it at any point. It naturally acts as a broadcast channel during a protocol, and we therefore sometimes write *broadcasting* when we mean that a party is writing to $\mathcal{F}_{\mathsf{GBB}}$. We use the formulation of [17] to model our global random oracle.

In their approach, the authors only let functionalities (machines with $\mathsf{pid} = \bot$) program the oracle. The calling functionality can initially be assigned control of

the oracle and is then referred to as the controller of the session. Only registered controllers of $\mathsf{sid}$ can program points that have not been queried before. In this formulation of the global random oracle the programming access is controlled by the calling functionalities. To not further complicate our functionalities we model this access structure as a macro MacroPrg described in Figure 4. Throughout the paper, we implicitly assume that every functionality first invokes **Init Control** and performs programming to $\mathcal{F}_{\mathsf{GpRO}}$ via the predefined macro. For clarity and brevity, this is not repeated in the description of individual functionalities.

**$\mathcal{F}_{\mathsf{GBB}}$ (global)**

$\mathcal{F}_{\mathsf{GBB}}$ is parameterized by $\mathsf{pid}^{\mathsf{sid}} = \mathcal{P}_1, \ldots, \mathcal{P}_n$. Only parties $\mathcal{P} \in \mathsf{pid}^{\mathsf{sid}}$ can write to $\mathcal{F}_{\mathsf{GBB}}$ for that session. Any party as well as any functionality can read all previously posted messages.

**Broadcast** On input $(\mathsf{Broadcast}, \mathsf{sid}, m)$ from $\mathcal{P}_i$:

1. If this is the first activation initialize $\mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$ and set $\mathsf{bid}^{\mathsf{sid}} = 0$.
2. If $\mathcal{P}_i$ in $\mathsf{pid}^{\mathsf{sid}}$, store the tuple $(\mathsf{bid}^{\mathsf{sid}}, \mathcal{P}_i, m)$ in $\mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$. Increase $\mathsf{bid}^{\mathsf{sid}}$ by 1 and send $(\mathtt{Broadcast}, m, \mathsf{bid}^{\mathsf{sid}})$ to every $\mathcal{P}_{j \neq i} \in \mathsf{pid}^{\mathsf{sid}}$.

**Verify** On input $(\mathsf{Verify}, \mathsf{sid})$ from any party return $\mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$.

Fig. 2: Ideal global bulletin board functionality.

**$\mathcal{F}_{\mathsf{GpRO}}$ (global)**

$\mathcal{F}_{\mathsf{GpRO}}$ interacts with a set of parties. For each session $\mathcal{F}_{\mathsf{GpRO}}$ can be controlled by a party with $\mathsf{sid} = \bot$ (functionalities) which has previously been assigned control. Let $\mathtt{mp}_{\mathrm{ctrl}}$ be an initially empty map storing controllers of each session and $\mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$ be an initially empty list for each $\mathsf{sid}$. $\mathcal{F}_{\mathsf{GpRO}}$ samples from a finite space $\mathcal{T}$ whose description length is polynomial in the security parameter.

**Assign control** Upon receiving $(\mathsf{Control}, \mathsf{sid})$ from a party $\mathcal{P}_i$ with $\mathsf{pid} = \bot$, add $(\mathsf{sid}, \mathcal{P}_i)$ to $\mathtt{mp}_{\mathrm{ctrl}}$ and return $\mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$.

**Query** On input $(\mathsf{Query}, \mathsf{sid}, m, \mathcal{T})$ from a party $\mathcal{P}_i$.

1. If $\mathcal{P}'$ is registered as the controller of this $\mathsf{sid}$ in $\mathtt{mp}_{\mathrm{ctrl}}$, output $(\mathsf{ProgramReq}, \mathsf{sid}, \mathcal{P}, m, \mathcal{T})$ to $\mathcal{P}'$.
2. Upon response $(\mathsf{Program}, \mathsf{sid}, m, \mathcal{T}, h)$ and if there is no record of the form $(m, \mathcal{T}, \cdot) \in \mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$, then add $(m, \mathcal{T}, h)$ to $\mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$.
3. Find $(m, \mathcal{T}, h) \in \mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$ if it is absent, generate $h \leftarrow\!\!\$\ \mathcal{T}$. Add the tuple to $\mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$ and send $(\mathsf{Query}, \mathsf{sid}, h)$ to the caller.

Fig. 3: Global Random Oracle functionality $\mathcal{F}_{\mathsf{GpRO}}$ following [17].

For $\mathcal{F}_{\mathsf{GpRO}}$ we use the following, well-known fact (see e.g. [5]):

**MacroPrg**

If any of the processes return $\perp$ abort the execution inside the calling functionality. Every functionality keeps track of initially empty map $\mathtt{li}_{\mathrm{H}}^{\mathsf{sid}}$. We only allow programming from the ideal process adversary ($\mathcal{S}$) with same $\mathsf{sid}$ as the calling functionality.

**Init control** send $(\mathsf{Control}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and wait for its response $m$. If $m = (\textsc{Ok}, Hist)$ and $Hist \neq \perp$ output $\perp$.

**Program** On input $(\mathsf{ProgramReq}, \mathsf{sid}, m, \mathcal{T})$ from $\mathcal{F}_{\mathsf{GpRO}}$. Send $(\mathsf{Program}, \mathsf{sid}, m, \mathcal{T}, h)$ to $\mathcal{S}$ and wait for its response $h$. Forward $(\mathsf{Program}, \mathsf{sid}, m, \mathcal{T}, h))$ to $\mathcal{F}_{\mathsf{GpRO}}$.

Fig. 4: Macro for programming the random oracle

**Proposition 1 (Random oracle graph game).** *Let $H\colon \{0,1\}^* \to \mathcal{T}$ be a random oracle and consider following game with an adversary $\mathcal{A}$. The game keeps track of a directed graph $G = (V, E)$, initially empty, and proceeds as follows: The adversary can (repeatedly) query $H$ at some $x$. Let $H(x) = y$.*

- *If $y \in V$ but there is no edge $(x', y) \in E$, then the adversary wins. (Preimage)*
- *If an edge $(x', y) \in E$ exists with $x' \neq x$, then the adversary wins. (Collision)*
- *Else, add nodes $x$ and $y$ to $V$ and add an edge $e = (x, y)$ from $x$ to $y$ to $E$.*

*Let $\mathcal{A}$ be an adversary which makes at most $Q$ queries to $H$. Then the probability that $\mathcal{A}$ wins is bounded by $Q^2/|\mathcal{T}|$.*

In addition, our protocols also use an Oblivious Linear Evaluation (OLE) functionality (see Figure 5). There exists a variety of protocols to efficiently implement this primitive, such as [23,24]. Protocols based on Pseudorandom Correlation Generators have been proposed, following the breakthrough work of [15]. They implement OLE with sublinear communication (in the correlation length $\ell$) where the shares of honest parties are pseudorandom.

**$\mathcal{F}_{\mathsf{OLE}}$**

$\mathcal{F}_{\mathsf{OLE}}$ interacts between two parties $\mathcal{P}_{\mathrm{S}}$ and $\mathcal{P}_{\mathrm{R}}$. The functionality is parameterized by the finite field $\mathbb{F}$ and output length $\ell$.

**Extend** On input $(\mathsf{Extend}, \mathsf{sid}, \mathbf{u})$ from $\mathcal{P}_{\mathrm{S}}$ and input $(\mathsf{Extend}, \mathsf{sid}, \mathbf{x})$ from $\mathcal{P}_{\mathrm{R}}$ where $\mathbf{u}, \mathbf{x} \in \mathbb{F}^{\ell}$.

1. If $\mathcal{P}_{\mathrm{R}}$ is corrupt wait for $\mathcal{A}$ to send $\mathbf{v}$. Otherwise sample $\mathbf{v} \leftarrow \mathbb{F}^{\ell}$.
2. If $\mathcal{P}_{\mathrm{S}}$ is honest, compute $\mathbf{w} = \mathbf{u} \cdot \mathbf{x} + \mathbf{v}$. If $\mathcal{P}_{\mathrm{S}}$ is corrupt wait for $\mathcal{A}$ to send $\mathbf{w}$ and recompute $\mathbf{v}$ accordingly.
3. Output $\mathbf{w}$ to $\mathcal{P}_{\mathrm{S}}$ and $\mathbf{v}$ to $\mathcal{P}_{\mathrm{R}}$.

Fig. 5: OLE functionality $\mathcal{F}_{\mathsf{OLE}}$.

### 2.3 VOLEs and VOLE-based MACs

Given a field $\mathbb{F}$, a VOLE correlation of length $\ell$ is a pair of random variables $(\boldsymbol{w}, \boldsymbol{u}) \in \mathbb{F}^\ell \times \mathbb{F}^\ell$ and $(\boldsymbol{v}, \Delta) \in \mathbb{F}^\ell \times \mathbb{F}$, subject to the constraint that $w_i = v_i + u_i \cdot \Delta$. The correlation exists between two parties, the sender $\mathcal{P}_\mathrm{S}$ and receiver $\mathcal{P}_\mathrm{R}$. $\mathcal{P}_\mathrm{S}$ is given $(\boldsymbol{w}, \boldsymbol{u})$, while $\mathcal{P}_\mathrm{R}$ learns $(\boldsymbol{v}, \Delta)$. It is an instantiation of the correlation generated by $\mathcal{F}_{\mathsf{OLE}}$ where all $x_i$ are $\Delta$.

VOLE can be used to build a simple, information-theoretic MAC scheme with homomorphic properties for use in MPC protocols, such as described in [13]. There, the secret held by the sender is $\mathbf{u}$ while the MACs are defined by $\mathbf{w}$. $\Delta, \mathbf{v}$ are the keys held by the verifier. To see that the scheme is linearly homomorphic, observe that for any $\alpha, \delta \in \mathbb{F}$ and VOLE correlations $w_1 = v_1 + u_1\Delta$ and $w_2 = v_2 + u_2\Delta$, it holds that

$$\underbrace{\alpha w_1 + w_2}_{\text{computed by } \mathcal{P}_\mathrm{S}} = \underbrace{\alpha v_1 + v_2 - \Delta\delta}_{\text{computed by } \mathcal{P}_\mathrm{R}} + \underbrace{(\alpha u_1 + u_2 + \delta)}_{\text{computed by } \mathcal{P}_\mathrm{S}} \Delta$$

is a valid VOLE correlation for the secret $\alpha u_1 + u_2 + \Delta$ whose MACs and keys can be locally computed by $\mathcal{P}_\mathrm{S}, \mathcal{P}_\mathrm{R}$ respectively.

To open a secret $u$, the sender sends both $u, w$ to $\mathcal{P}_\mathrm{R}$, which in turn checks that $w = u \cdot \Delta + v$. The MAC $w$ cannot be forged for a different secret $u' \neq u$ with probability larger than $1/|\mathbb{F}|$ (see e.g. [9, Section 3.1] for a proof):

**Lemma 1.** *Let $v, \Delta$ be a VOLE commitment to $u$ with MAC $w$. Assume that $\mathcal{P}_S$ sends $u', w' \in \mathbb{F}$ to $\mathcal{P}_R$ such that $u \neq u'$. Let $\mathcal{P}_R$ accept if $w' = u'\Delta + v$, and otherwise reject. Then $\mathcal{P}_R$ accepts with probability at most $1/|\mathbb{F}|$.*

## 3 Formalizing Auditable MPC

We now recap the UC formalization of Publicly Verifiable (Auditable) Secure Multiparty Computation as first presented in [8].

In Figure 6 we present the ideal functionality $\mathcal{F}_{\mathsf{Online}}$. In $\mathcal{F}_{\mathsf{Online}}$ each party can input private values $x^i \in \mathbb{F}$ for a fixed field $\mathbb{F}$ and compute a predetermined circuit using these values. The circuit can be built from multiplication and linear gates corresponding to the same operations in $\mathbb{F}$. Moreover, the parties $\mathcal{P}$ can, upon agreement, open the outputs of the computation. In addition, an auditor $\mathcal{P}_\mathrm{A}$ is an additional party that can interact with the functionality and verify the correctness of the computation output $y$ after it is opened. The auditor will, on input $y$, output accept if the circuit $\mathcal{C}$ on input $x^1, \ldots, x^n$ outputs $y$. If the value $y$ is not identical to the output of $\mathcal{C}$, the auditor will output reject.

*Parties.* The functionality interacts with a set of $n$ parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ that provide inputs and evaluate the circuit $\mathcal{C}$. The adversary $\mathcal{A}$ can corrupt arbitrarily many of the parties. If at least one party is honest correctness and input privacy

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

$\mathcal{F}_{\mathsf{Online}}$ interacts with an adversary $\mathcal{A}$, a set of computation parties $\mathcal{P}$ and an auditor $\mathcal{P}_{\mathrm{A}}$. Let $\mathbb{F}$ be a field.

**Initialize** On input $(\mathsf{Init}, \mathsf{sid}, \mathcal{C})$ from all parties in $\mathcal{P}$, where $\mathcal{C}$ is a circuit with $n$ inputs, one output, and addition and multiplication gates over $\mathbb{F}$:
1. $\mathcal{A}$ sends the set of corrupted parties $\tilde{\mathcal{P}} \subseteq \mathcal{P}$.

**Input** On input $(\mathsf{Input}, \mathsf{sid}, i, \mathsf{id}_x, x)$ from $\mathcal{P}_i$ with $x \in \mathbb{F}$ and $(\mathsf{Input}, \mathsf{sid}, i, \mathsf{id}_x)$ from all other parties.
1. Store $(\mathsf{id}_x, x)$ and ignore all messages $(\mathsf{Input}, \mathsf{sid})$ with the same $\mathsf{id}_x$.
2. If all parties are corrupted, send $(\mathsf{Input}, \mathsf{sid}, i, \mathsf{id}_x, x)$ to $\mathcal{A}$.

**Compute** On input $(\mathsf{Compute}, \mathsf{sid})$ from all parties $\mathcal{P}_i$:
1. Assign inputs to input gates of $\mathcal{C}$. If some input gates are unassigned, output $(\mathsf{Abort}, \mathsf{sid})$.
2. Compute $y_c = \mathcal{C}(x_1, \ldots, x_n)$.
3. Send $y_c$ to $\mathcal{A}$ and wait for $\mathcal{A}$ to respond with $y^*$. If $|\tilde{\mathcal{P}}| = n$ all values $y^* \in \mathbb{F} \cup \{\perp\}$ are valid. If $|\tilde{\mathcal{P}}| < n$ only $y^* \in \{y_c, \perp\}$ is valid.
4. Send $(\mathsf{Output}, \mathsf{sid}, y^*)$ to all $\mathcal{P}_i$ if $y^*$ is valid, and $(\mathsf{Abort}, \mathsf{sid})$ otherwise.

**Audit** On input $(\mathsf{Audit}, \mathsf{sid}, y)$ from $\mathcal{P}_{\mathrm{A}}$ and if **Compute** has finished:
1. If $y = y_c = y^*$ send $(\mathsf{Accept}, \mathsf{sid})$ to $\mathcal{P}_{\mathrm{A}}$.
2. If $y^* = \perp$ send $(\mathsf{AuditNotPossible}, \mathsf{sid})$ to $\mathcal{P}_{\mathrm{A}}$.
3. If $y \neq y_c$ or $y \neq y^*$ send $(\mathsf{Reject}, \mathsf{sid})$ to $\mathcal{P}_{\mathrm{A}}$.

Fig. 6: Functionality $\mathcal{F}_{\mathsf{Online}}$ of Auditable MPC.

are guaranteed. If all parties are dishonest only correctness can be guaranteed and the auditor $\mathcal{P}_{\mathrm{A}}$ can verify the output after the computation of $\mathcal{C}$ is finished. We model $\mathcal{P}_{\mathrm{A}}$ as an incorruptible party.

*Realizing* $\mathcal{F}_{\mathsf{Online}}$. To maintain a state that is shared by all protocols we use session identifiers $\mathsf{sid}$ throughout our construction. When realizing $\mathcal{F}_{\mathsf{Online}}$, all steps of the computation are send to the bulletin board, such that $\mathcal{P}_{\mathrm{A}}$ can later get the transcript of the computation. We present the protocol in Section 6.

## 4 One-time Homomorphic Commitments

In this section, we build a one-time homomorphic commitment scheme from a weak form of VOLE. Namely, we use a special VOLE functionality, whose ideas were introduced in [7], with publicly revealed $\Delta$ to implement VOLE-based commitments. The commitment functionality $\mathcal{F}_{\mathsf{HOTCom}}$ that we realize is described in Figures 7 and 8.

We now describe the functionality in more detail. The functionality $\mathcal{F}_{\mathsf{HOTCom}}$ is a multi-receiver commitment scheme, run between a sender $\mathcal{P}_{\mathrm{S}}$, a set of receivers $\mathcal{P}_{\mathrm{R}}$ and an auditor $\mathcal{P}_{\mathrm{A}}$. After initialization using **Init**, the sender may create mc commitments in multiple rounds using **Commit**. Linear operations can then be applied to previously committed values using **Add** and **Linear**. Commitments

**$\mathcal{F}_{\mathsf{HOTCom}}$ (Part 1)**

$\mathcal{F}_{\mathsf{HOTCom}}$ interacts with the sender $\mathcal{P}_{\mathrm{S}}$, a set of receivers $\mathcal{P}_{\mathrm{R}}$, an auditor $\mathcal{P}_{\mathrm{A}}$ and an adversary $\mathcal{A}$ who is allowed to corrupt every party except $\mathcal{P}_{\mathrm{A}}$. If $\mathcal{A}$ or a party sends $(\mathsf{Abort}, \mathsf{sid})$ it halts.
The functionality is parameterized by a field $\mathbb{F}_p$. Let $\mathtt{fl}_{\mathrm{open}}$ be a flag that is initially 0 and $\mathtt{li}_{\mathrm{gen}}, \mathtt{li}_{\mathrm{open}}, \mathtt{li}_{\mathrm{chal}}$ be lists and $\mathtt{mp}_{\mathrm{com}}, \mathtt{mp}_{\mathrm{chal}}$ be maps - all are initially empty. Challenges are generated from a finite set $\mathcal{T}$.

**Init** Upon input $(\mathsf{Init}, \mathsf{sid}, \mathrm{mc})$ from $\mathcal{P}_{\mathrm{S}}$ set $\mathtt{li}_{\mathrm{gen}} := \{\mathsf{cid}_1, \ldots, \mathsf{cid}_{\mathrm{mc}}\}$ and output $(\mathsf{Init}, \mathsf{sid}, \mathrm{mc})$ to all parties in $\mathcal{P}_{\mathrm{R}}$ and $\mathcal{A}$. Let $\mathcal{U}$ be the set of all $\mathbb{F}_p$-linear functions in $\mathsf{cid}_1, \ldots, \mathsf{cid}_{\mathrm{mc}}$.

**Commit** On input $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{cid}_j, m_j)_{j\in[k]})$ from $\mathcal{P}_{\mathrm{S}}$, if $\mathtt{fl}_{\mathrm{open}} = 0$, $\{\mathsf{cid}_j\}_{j\in[k]} \subseteq \mathtt{li}_{\mathrm{gen}}$, $m_j \in \mathbb{F}_p$ and $(\mathsf{cid}_j, *) \notin \mathtt{mp}_{\mathrm{com}}$ for all $j \in [k]$:

1. Let $(\mathsf{state}, *)$ be the last item on $\mathtt{li}_{\mathrm{chal}}$ or $\mathsf{state} = \bot$ if it is empty. If $\mathcal{P}_{\mathrm{S}}$ is corrupted, wait for message $(\mathsf{ChalSel}, \mathsf{sid}, c)$ and let $\mathbf{ch}$ be as in $(\mathtt{li}_{\mathrm{gen}} \| (\mathsf{state}, (\mathsf{cid}_j, m_j)_{j\in[k]}) \| c, \mathbf{ch}) \in \mathtt{mp}_{\mathrm{chal}}$. If sender is honest, set $\mathbf{ch} \xleftarrow{\$} \mathcal{T}$. Add $(\mathsf{state} \| (\mathsf{cid}_j, m_j)_{j\in[k]}, \mathbf{ch})$ to $\mathtt{li}_{\mathrm{chal}}$.
2. Add the value $(\mathsf{cid}_j, (m_j, \mathsf{id}_j))$ for all $j \in [k]$ to $\mathtt{mp}_{\mathrm{com}}$ where $\mathsf{id}_j$ is the identity function in $\mathcal{U}$.
3. Send $(\mathsf{Receipt}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, \mathbf{ch})$ to all $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$, $\mathcal{P}_{\mathrm{S}}$ and $\mathcal{A}$.

**Challenge Test** On input $(\mathsf{TestChallenge}, \mathsf{sid}, \mathrm{mc}, L)$ from $\mathcal{A}$ where $L = (\mathsf{cid}_j, m_j)_{j\in[k]}$ is a list with elements with each $\mathsf{cid}_j \in T = \{\mathsf{cid}_1, \ldots, \mathsf{cid}_{\mathrm{mc}}\}$ and if $\mathcal{P}_{\mathrm{S}}$ is corrupted:

1. If $(T \| L \| i, *) \in \mathtt{mp}_{\mathrm{chal}}$ set $c = i + 1$ for the largest $i$, otherwise $c = 0$.
2. Sample $\mathbf{ch} \xleftarrow{\$} \mathcal{T}$, add $(T \| L \| c, \mathbf{ch})$ to $\mathtt{mp}_{\mathrm{chal}}$ and send $(\mathbf{ch}, c)$ to $\mathcal{A}$.

**Hint** On input $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, i)$ from $\mathcal{P}_{\mathrm{S}}$, if $(\mathsf{cid}_j, (m_j, *)) \in \mathtt{mp}_{\mathrm{com}}$ for every $j \in [k]$, $\mathtt{fl}_{\mathrm{open}} = 0$ and $i \in \{0, \ldots, |\mathcal{P}_{\mathrm{R}}|\}$:

1. If $\mathcal{P}_{\mathrm{S}}$ is corrupt, wait for input $(\mathsf{Hint}, \mathsf{sid}, (x_j)_{j\in[k]})$ from $\mathcal{A}$ where $x_j \in \mathbb{F}_p$. Otherwise set $x_j = m_j$.
2. For each $j \in [k]$ if $x_j := m_j$ add $(\mathsf{cid}_j, \neg\mathtt{delay}, x_j, 1, i)$ to $\mathtt{li}_{\mathrm{open}}$, otherwise add $(\mathsf{cid}_j, \neg\mathtt{delay}, x_j, 0, i)$.
3. If $i = 0$ send $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j, x_j)_{j\in[k]})$ to all $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$ and $\mathcal{A}$.
4. If $i \neq 0$ send $(\mathsf{PrivHint}, \mathsf{sid}, (\mathsf{cid}_j, x_j)_{j\in[k]})$ to $\mathcal{P}_i$.

**Delayed Hint** On input $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$ from $\mathcal{P}_{\mathrm{S}}$, if $(\mathsf{cid}_j, (m_j, *)) \in \mathtt{mp}_{\mathrm{com}}$ for every $j \in [k]$ and if $\mathtt{fl}_{\mathrm{open}} = 0$:

1. If $\mathcal{P}_{\mathrm{S}}$ is corrupt, wait for input $(\mathsf{Hint}, \mathsf{sid}, (x_j)_{j\in[k]})$ from $\mathcal{A}$ where $x_j \in \mathbb{F}_p$. Otherwise set $x_j := m_j$.
2. For every $j \in [k]$ if $x_j = m_j$ add $(\mathsf{cid}_j, \mathtt{delay}, x_j, 1)$ to $\mathtt{li}_{\mathrm{open}}$, otherwise add $(\mathsf{cid}_j, \mathtt{delay}, x_j, 0)$.
3. Send $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$ to all parties in $\mathcal{P}_{\mathrm{R}}$.

Fig. 7: Ideal functionality for one-time homomorphic commitments.

and linear operations can be done until the sender calls **Open**. Thereafter, these operations are not permitted anymore as a special flag $\mathtt{fl}_{\mathrm{open}}$ is then set. Commitments and operations on them are internally stored in a special map $\mathtt{mp}_{\mathrm{com}}$

$\mathcal{F}_{\mathsf{HOTCom}}$ **(Part 2)**

**Add** On input $(\mathsf{Add}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[3]})$ from $\mathcal{P}_S$ and if $\mathtt{fl}_{\text{open}} = 0$, check if $(\mathsf{cid}_3, *) \notin \mathtt{mp}_{\text{com}}$, $\mathsf{cid}_3 \notin \mathtt{li}_{\text{gen}}$ and $\{(\mathsf{cid}_1, (m_1, f_1)), (\mathsf{cid}_2, (m_2, f_2))\} \subseteq \mathtt{mp}_{\text{com}}$:

1. Store tuple $(\mathsf{cid}_3, (m_1 + m_2, f_1 + f_2))$ in $\mathtt{mp}_{\text{com}}$.
2. Send $(\mathsf{Add}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[3]})$ to all $\mathcal{P}_i \in \mathcal{P}_R$ and $\mathcal{A}$.

**Linear** On input $(\mathsf{Linear}, \mathsf{sid}, \mathsf{cid}_1, \mathsf{cid}_2, \alpha, \delta)$ from $\mathcal{P}_S$ and if $\mathtt{fl}_{\text{open}} = 0$, if $(\mathsf{cid}_2, *) \notin \mathtt{mp}_{\text{com}}$,$\mathsf{cid}_2 \notin \mathtt{li}_{\text{gen}}$ and $(\mathsf{cid}_1, (m_1, f_1)) \in \mathtt{mp}_{\text{com}}$:

1. Store $(\mathsf{cid}_2, \alpha \cdot m_1 + \delta, \alpha \cdot f_1 + \delta)$ in $\mathtt{mp}_{\text{com}}$.
2. Send $(\mathsf{Linear}, \mathsf{sid}, \mathsf{cid}_1, \mathsf{cid}_2, \alpha, \delta)$ to all $\mathcal{P}_i \in \mathcal{P}_R$ and $\mathcal{A}$.

**Open** On input $(\mathsf{Open}, \mathsf{sid})$ from $\mathcal{P}_S$ and if $\mathtt{fl}_{\text{open}} = 0$: Set $\mathtt{fl}_{\text{open}} \leftarrow 1$ and send $(\mathsf{Open}, \mathsf{sid})$ to all parties.

**Delayed Open** On input $(\mathsf{Open}, \mathsf{sid})$ from $\mathcal{P}_S$ and if $\mathtt{fl}_{\text{open}} = 1$:

1. Let $L := \{(\mathsf{cid}_j, x_j) \mid (\mathsf{cid}_j, \mathtt{delay}, x_j, b_j) \in \mathtt{li}_{\text{open}}\}$.
2. Replace each $(\mathsf{cid}_j, \mathtt{delay}, x_j, b_j) \in \mathtt{li}_{\text{open}}$ with $(\mathsf{cid}_j, \neg\mathtt{delay}, x_j, b_j, 0)$.
3. Set $\mathtt{fl}_{\text{open}} \leftarrow 2$ and send $(\mathsf{Open}, \mathsf{sid}, L)$ to each party in $\mathcal{P}_R$.

**Verify** On input $(\mathsf{Verify}, \mathsf{sid}, (\mathsf{cid}_j, m_j)_{j\in[k]})$ from any party $\mathcal{P}_i \in \mathcal{P}_R$ and if $\mathtt{fl}_{\text{open}} \neq 0$:

- Return $(\mathsf{Verified}, \mathsf{sid})$ if $(\mathsf{cid}_j, \neg\mathtt{delay}, m_j, 1, 0) \in \mathtt{li}_{\text{open}}$ or $(\mathsf{cid}_j, \neg\mathtt{delay}, m_j, 1, i) \in \mathtt{li}_{\text{open}}$ for every $j \in [k]$
- Otherwise return $(\mathsf{VerifyFailed}, \mathsf{sid})$.

**Audit** On input $(\mathsf{Audit}, \mathsf{sid}, (\mathsf{cid}_j, f_j)_{j\in[k]})$ by $\mathcal{P}_A$, if $\mathsf{cid}_j \in \mathcal{U}$ and $\mathtt{fl}_{\text{open}} = 2$:

1. Let $L := \{(\mathsf{cid}_j, m_j) \mid (\mathsf{cid}_j, \neg\mathtt{delay}, m_j, 1, 0) \in \mathtt{li}_{\text{open}}\}$.
2. Return $(\mathsf{Audit}, \mathsf{sid}, L)$ if $(\mathsf{cid}_j, \neg\mathtt{delay}, m_j, 1, 0) \in \mathtt{li}_{\text{open}}$ and $(\mathsf{cid}_j, (m_j, f_j))) \in \mathtt{mp}_{\text{com}}$, for every $j \in [k]$. Otherwise return $(\mathsf{AuditFailed}, \mathsf{sid})$.

**Challenges** On input $(\mathsf{Challenges}, \mathsf{sid})$ from any party or $\mathcal{A}$ output $L = (((\mathsf{cid}_j), \mathbf{ch}_k) \mid ((\mathsf{cid}_j, m_j), \mathbf{ch}_k) \in \mathtt{li}_{\text{chal}})$.

Fig. 8: Ideal functionality for one-time homomorphic commitments, continued.

based on unique commitment IDs $\mathsf{cid}$. Some of the commitment IDs are reserved for using **Commit** and stored in a list $\mathtt{li}_{\text{gen}}$.

Instead of multiple rounds of openings of commitments, the functionality permits so-called *hints*. A hint allows an honest party to reveal a committed value in a verifiable way, while a dishonest sender can change the hint outputs at will. However, since hints are verifiable, such a change to the outputs is visible after **Open** is called. Hints can be public and private, meaning that output can either be opened to an individual receiver or all receivers in a verifiable way. *Private hints* are only verifiable by the receiver and no other party.

A sender can also *send hints to the future* by using **Delayed Hints** before calling **Open**. Revealing these hints through **Delayed Open** can only be used after **Open**. The (dishonest) sender can always decide not to reveal such a hint at this point, but it cannot change the value that is revealed anymore. Also, once such a delayed hint is revealed it is verifiable by all parties, just as regular

hints. $\mathcal{F}_{\mathsf{HOTCom}}$ keeps track of hints and openings in the list $\mathtt{li}_{\mathrm{open}}$, including information about delayed hints now being visible or receivers of hints.

*Audits.* The auditor $\mathcal{P}_{\mathrm{A}}$ acts as an additional receiver that is incorruptible. It can, however, not obtain private hints. In comparison to the other parties, it can only verify hints after **Delayed Open** is called. At that point, it can verify both regular and delayed hints.

*Generating challenges.* The protocol implementing $\mathcal{F}_{\mathsf{HOTCom}}$ consists of random oracle calls and messages sent to a bulletin board $\mathcal{F}_{\mathsf{GBB}}$. As such, this permits generating unpredictable challenges (to be used by protocols using $\mathcal{F}_{\mathsf{HOTCom}}$ as a building block) using the Fiat-Shamir paradigm where the messages on the Bulletin Board are hashed. However, this allows a corrupt sender, ahead of time, to *simulate* protocol transcripts before posting to $\mathcal{F}_{\mathsf{GBB}}$, thus seeing what Fiat-Shamir challenges the simulated transcripts lead to. To capture this advantage, we additionally allow the attacker to use the functionality such that it shows which challenges its input choices generate. That is, we *explicitly* model challenge generation into the functionality, as well as explicit challenge testing which the adversary can do using **Challenge Test**.

During **Commit**, the dishonest sender can then pick any such tested challenge that it observed before. Based on the state, challenges can then be read by other parties such as the auditor using **Challenges**. The challenge space (essentially, the output space of the underlying random oracle) is defined by $\mathcal{T}$. To ensure consistency, mappings between functionality states and generated challenges are stored in a map $\mathtt{mp}_{\mathrm{chal}}$. The different challenges that can be derived publicly given the current and previous states of the functionality are stored in $\mathtt{li}_{\mathrm{chal}}$.

### 4.1 Weak VOLE

To implement $\mathcal{F}_{\mathsf{HOTCom}}$ we use a *weak* form of VOLE, which we describe in $\mathcal{F}_{\mathsf{WVOLE}}$ in Figure 9. It implements a form of VOLE correlation generation, described in Section 2.3 to allow linear commitments. However, it does not generate the correlation shares of the sender and the receiver simultaneously. Instead, initially $\mathcal{P}_{\mathrm{S}}$ learns its shares of the correlation, as well as $\Delta$. The choice of $\Delta$ is dependent on the message that it intends to send on the broadcast channel, which is modeled by the additional input $m$. $\mathcal{P}_{\mathrm{R}}$ learns $\Delta, m$, but initially not $\mathbf{Q}$. However, once the parties enter **Deliver** then $\mathbf{Q}$ is forwarded to $\mathcal{P}_{\mathrm{R}}$. Note that every choice of $m$ leads to a different $\Delta$, and an adversary can test which $m$ generates which $\Delta$ without generating interaction via **Delta Test**. This reflects that the implementation later generates $\Delta$ as the output of a hash function evaluated on the protocol transcript and $m$. The functionality stores the $(m, \Delta)$ that is chosen by the adversary in the flag $\mathtt{fl}_{\Delta}$. The functionality does not keep any secret from the sender if it is dishonest. In addition, $\mathcal{F}_{\mathsf{WVOLE}}$ allows an auditor to obtain the same information as $\mathcal{P}_{\mathrm{R}}$ at any later point. It is precisely the use of $m$ that ties $\mathbf{Q}, \Delta$ to an external context (such as other messages on a broadcast channel) making audit meaningful even if both $\mathcal{P}_{\mathrm{S}}, \mathcal{P}_{\mathrm{R}}$ are corrupted.

To allow using $\mathcal{F}_{\mathsf{WVOLE}}$ in a Fiat-Shamir context by other protocols, the functionality also generates an additional challenge **ch** once $\mathbf{U}, \mathbf{V}$ are fixed by the sender *but before the VOLE challenge $\Delta$ is chosen*. This allows higher-level protocols to derive an unpredictable value **ch** once inputs are fixed. This is in line with an actual implementation where $\mathbf{U}, \mathbf{V}$ can already be extracted from random oracle-based commitments before the opening takes place later, for which $\Delta$ is necessary. The challenge **ch** will later be computed by hashing RO commitments containing $\mathbf{U}, \mathbf{V}$. Since the adversary could simulate how multiple choices of $\mathbf{U}, \mathbf{V}$ and commitments thereof generate different hash function outputs and thus **ch** values, we allow the adversary to test output challenges using **Challenge Test** before inputting them to the functionality. As all this information is available as a public transcript and **ch** can also be derived by other parties, the functionality allows third parties to generate challenges using **Challenge**. As in $\mathcal{F}_{\mathsf{HOTCom}}$, we use a map $\mathtt{mp}_{\mathrm{chal}}$ to keep track of all challenges generated to the adversary and store the ultimately chosen challenge in the flag $\mathtt{fl}_{\mathrm{ch}}$.

A previous (interactive) version of $\mathcal{F}_{\mathsf{WVOLE}}$ was presented in [7], where the authors also show how to instantiate it. We adjust their protocols to fit our functionality. The asynchronous generation of $\Delta, \mathbf{Q}$ can easily be realized by running the original protocol from [7], but where $\Delta$ is now chosen based on a random oracle evaluated on the sender's message. $\mathcal{P}_{\mathrm{A}}$'s operations can be permitted by letting the parties communicate via a broadcast channel. This way, $\mathcal{P}_{\mathrm{A}}$ can obtain $\Delta, \mathbf{Q}$ and other information the same way as $\mathcal{P}_{\mathrm{R}}$ does in the protocol. These modifications to the protocol and the security proof of [7] can be found in Appendix D.

### 4.2 From weak VOLE to single-receiver one-time commitments

We will now describe a protocol that implements $\mathcal{F}_{\mathsf{HOTCom}}$ in the $\mathcal{F}_{\mathsf{WVOLE}}$-hybrid model for $|\mathcal{P}_{\mathrm{R}}| = 1$, i.e. for one receiver, working over $\mathbb{F}_p$ for large values of $p$. For small $p$, we refer to Appendix A.

*How to set up commitments.* Between sender $\mathcal{P}_{\mathrm{S}}$ and receiver $\mathcal{P}_{\mathrm{R}}$, we assume the existence of one instance of $\mathcal{F}_{\mathsf{WVOLE}}$ where $\mathcal{P}_{\mathrm{S}}$ is the sender and $\mathcal{P}_{\mathrm{R}}$ acts as receiver. The parties wish to commit to at most mc messages, where $k_{\mathcal{C}}$ is the dimension of an underlying linear code $\mathcal{C}$ and we set $\ell = \lceil \mathrm{mc}/k_{\mathcal{C}} \rceil$. The parties run **Init** on $\mathcal{F}_{\mathsf{WVOLE}}$ and $\mathcal{P}_{\mathrm{S}}$ learns $\mathbf{U}, \mathbf{V}$. Each of these matrices have $2\ell$ rows, where we use $\mathbf{U}$ to define the commitments to messages and $\mathbf{V}$ to perform consistency checks. We refer to the commitments from $\mathbf{U}$ as "level 1" VOLEs, and the VOLEs in the consistency check (which additionally use $\mathbf{V}$) as "level 0" VOLEs.

More concretely, sender uses $\mathbf{U} \in \mathbb{F}_p^{2\ell \times k_{\mathcal{C}}}$ to define a "level 1" VOLE with a yet unknown challenge $\Delta_1$. The upper half of $\mathbf{U}$, i.e. the first $\ell$ rows, will be used for the message part of the VOLE (the degree-1 coefficient, which we call $\mathbf{U}_1$) commitments, while its lower half of $\mathbf{U}$ will act as the MAC (the degree 0 coefficient of each VOLE, also called $\mathbf{U}_2$). That means, the "level 1" VOLEs are

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

The functionality interacts with a sender $\mathcal{P}_{\mathrm{S}}$, a receiver $\mathcal{P}_{\mathrm{R}}$, an adversary $\mathcal{A}$, an auditor $\mathcal{P}_{\mathrm{A}}$ and the functionality $\mathcal{F}_{\mathsf{GpRO}}$. Upon initialization, $\mathcal{A}$ can corrupt any of $\mathcal{P}_{\mathrm{S}}, \mathcal{P}_{\mathrm{R}}$ or even both, but never $\mathcal{P}_{\mathrm{A}}$. $\mathcal{F}_{\mathsf{WVOLE}}$ is parametrized by a prime $p$ as well as an $[n_{\mathcal{C}}, k_{\mathcal{C}}, d_{\mathcal{C}}]_p$ linear code $\mathcal{C}$ over $\mathbb{F}_p$ with generator matrix $\mathbf{G} \in \mathbb{F}_p^{k_{\mathcal{C}} \times n_{\mathcal{C}}}$ and a set $[N] \subseteq \mathbb{F}_p$. Let $\mathcal{T}$ be a set of possible challenges and $\mathtt{fl}_{\mathrm{ch}}, \mathtt{fl}_{\Delta}$ be flags that are initially $\perp$. Let $\mathtt{mp}_{\mathrm{chal}}$ be a map that is initially empty.

**Init** Upon first input $(\mathsf{Init}, \mathsf{sid}, \ell)$ from $\mathcal{P}_{\mathrm{S}}$:

1. If $\mathcal{P}_{\mathrm{S}}$ is honest, then sample $\mathbf{U} \xleftarrow{\$} \mathbb{F}_p^{\ell \times k_{\mathcal{C}}}$, $\mathbf{V} \xleftarrow{\$} \mathbb{F}_p^{\ell \times n_{\mathcal{C}}}$, and $\mathbf{ch} \xleftarrow{\$} \mathcal{T}$, set $\mathtt{fl}_{\mathrm{ch}} \leftarrow (\ell, \mathbf{ch})$ and send $(\mathsf{Sender}, \mathsf{sid}, \mathbf{U}, \mathbf{V}, \mathbf{ch})$ to $\mathcal{P}_{\mathrm{S}}$.
2. If $\mathcal{P}_{\mathrm{S}}$ is corrupt, receive $(\mathsf{Sender}, \mathsf{sid}, \mathbf{U}, \mathbf{V}, c)$ from $\mathcal{A}$ where $(\ell \| \mathbf{U} \| \mathbf{V} \| c, \mathbf{ch}) \in \mathtt{mp}_{\mathrm{chal}}$. Then set $\mathtt{fl}_{\mathrm{ch}} \leftarrow (\ell, \mathbf{ch})$.
3. Finally, send $(\mathsf{Init}, \mathsf{sid}, \ell, \mathbf{ch})$ to $\mathcal{P}_{\mathrm{R}}$.

**Init Done** Upon first input $(\mathsf{InitDone}, \mathsf{sid}, m)$ from $\mathcal{P}_{\mathrm{S}}$ and if $\mathtt{fl}_{\mathrm{ch}} \neq \perp$:

1. Set $\Delta \leftarrow \mathtt{getDelta}(\mathsf{sid}, \ell, \mathbf{ch}, m)$ and $\mathtt{fl}_{\Delta} \leftarrow (m, \Delta)$.
2. Finally, send $(\mathsf{Done}, \mathsf{sid}, m, \Delta)$ to both $\mathcal{P}_{\mathrm{S}}$ and $\mathcal{P}_{\mathrm{R}}$.

**Deliver** Upon first input $(\mathsf{Deliver}, \mathsf{sid})$ from $\mathcal{P}_{\mathrm{S}}$ and if $\mathtt{fl}_{\Delta} \neq \perp$:

1. Compute $\mathbf{Q} := \mathbf{V} + \mathbf{U}\mathbf{G}\mathsf{diag}(\Delta)$ and send $(\mathsf{Corr}, \mathsf{sid}, \mathbf{Q})$ to $\mathcal{P}_{\mathrm{R}}$.

**Challenge Test** On input $(\mathsf{TestChallenge}, \mathsf{sid}, \ell, \mathbf{U}, \mathbf{V})$ from $\mathcal{A}$ where $\mathbf{U} \in \mathbb{F}_p^{\ell \times k_{\mathcal{C}}}$ and $\mathbf{V} \in \mathbb{F}_p^{\ell \times n_{\mathcal{C}}}$:

1. If $(\ell \| \mathbf{U} \| \mathbf{V} \| c, *) \in \mathtt{mp}_{\mathrm{chal}}$ then let $i = c + 1$ where $c$ is the highest such value. Otherwise, let $c = 0$.
2. Sample $\mathbf{ch} \xleftarrow{\$} \mathcal{T}$, store $(\ell \| \mathbf{U} \| \mathbf{V} \| c, \mathbf{ch})$ in $\mathtt{mp}_{\mathrm{chal}}$ and send $(c, \mathbf{ch})$ to $\mathcal{A}$.

**Delta Test** On input $(\mathsf{TestDelta}, \mathsf{sid}, \ell, \mathbf{ch}, m)$ from $\mathcal{A}$ where $\mathbf{ch} \in \mathcal{T}$ and $m \in \{0,1\}^*$: Return $\mathtt{getDelta}(\mathsf{sid}, \ell, \mathbf{ch}, m)$.

**Audit** Upon receiving $(\mathsf{Audit}, \mathsf{sid})$ from $\mathcal{P}_{\mathrm{A}}$:

- If no $(\mathsf{Corr}, \mathsf{sid})$ message was sent to $\mathcal{P}_{\mathrm{R}}$, then return $(\mathsf{AuditFailed}, \mathsf{sid})$.
- Otherwise return $(\mathsf{Corr}, \mathsf{sid}, \ell, \mathbf{ch}, m, \Delta, \mathbf{Q})$ where $\mathtt{fl}_{\mathrm{ch}} = (\ell, \mathbf{ch}), \mathtt{fl}_{\Delta} = (m, \Delta)$.

**Challenge** Upon receiving $(\mathsf{Challenge}, \mathsf{sid})$ from any party or $\mathcal{A}$ let $\mathtt{fl}_{\mathrm{ch}} = (\ell, \mathbf{ch})$ and return $(\mathsf{Challenge}, \mathsf{sid}, \mathbf{ch})$.

**Procedure** $\mathtt{getDelta}(\mathsf{sid}, \ell, \mathbf{ch}, m)$**:**

1. If $(\ell \| \mathbf{ch} \| m, \Delta') \in \mathtt{mp}_{\mathrm{chal}}$ then set $\Delta := \Delta'$
2. Otherwise sample $\Delta \xleftarrow{\$} [N]^{n_{\mathcal{C}}}$ and store $(\ell \| \mathbf{ch} \| m, \Delta)$ in $\mathtt{mp}_{\mathrm{chal}}$.
3. Return $\Delta$.

Fig. 9: Ideal Functionality $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ for auditable weak VOLE.

formed by the equation $\mathbf{U}_1 X + \mathbf{U}_2$ modulo $p$ where $X$ is the indeterminate, to be replaced later by $\Delta_1$. After **Init** terminates, the sender therefore holds the secrets and MACs of *random* level-1 VOLEs which it can use to commit.

For each linear operation on the level-1 VOLE commitments, sender and receiver keep track of the linear functions which they apply to the level-1 VOLE commitments generated as mentioned above. Given an input $m$ to be committed to and an unused index $(i, j) \in [\ell] \times [k_{\mathcal{C}}]$ in $\mathbf{U}_1$ (i.e. an unused level-1 VOLE),

**$\Pi^{1-1}_{\mathsf{HOTCommit}}$ (Part 1)**

The protocol is run by a sender $\mathcal{P}_{\mathrm{S}}$, a receiver $\mathcal{P}_{\mathrm{R}}$ and an auditor $\mathcal{P}_{\mathrm{A}}$. It is parametrized by a prime $p$ as well as the $[n_{\mathcal{C}}, k_{\mathcal{C}}, d_{\mathcal{C}}]_p$ linear code $\mathcal{C}$ over $\mathbb{F}_p$. We define the generator matrix $\mathbf{G} \in \mathbb{F}_p^{k_{\mathcal{C}} \times n_{\mathcal{C}}}$ for $\mathcal{C}$ as well as a set $S_\Delta \subseteq \mathbb{F}_p$. There exists an instance $\mathcal{F}_{\mathsf{WVOLE}}$ of $\mathcal{F}^{p,N,\mathcal{C}}_{\mathsf{WVOLE}}$ between $\mathcal{P}_{\mathrm{S}}$ and receiver $\mathcal{P}_{\mathrm{R}}$ which $\mathcal{P}_{\mathrm{A}}$ can also access. Moreover, all parties have access to a random oracle $\mathcal{F}_{\mathsf{GpRO}}$ and bulletin board $\mathcal{F}_{\mathsf{GBB}}$. Let $\mathcal{T}$ be a challenge space. In the following, when we say that a party aborts then it also broadcasts $(\mathsf{Abort}, \mathsf{sid})$.

Each party maintains initially empty lists $\mathtt{li}_{\mathrm{gen}}, \mathtt{li}_{\mathrm{chal}}$ and maps $\mathtt{mp}_{\mathrm{com}}, \mathtt{mp}_{\mathrm{open}}$. We denote by $\mathsf{f}_j \in \mathbb{F}_p[X_1, \ldots, X_{\ell k_{\mathcal{C}}}]$ the polynomial $\mathsf{f}_j(x_1, \ldots, x_{\ell k_{\mathcal{C}}}) = x_j$. For any degree-1 polynomial $f$ let $f^{(0)}$ denote the polynomial that consists of the degree-0 monomials of $f$ and $f^{(1)}$ be the sum of the degree-1 monomials of $f$.

**Init** On first input $(\mathsf{Init}, \mathsf{sid}, \mathrm{mc})$ to $\mathcal{P}_{\mathrm{S}}$:

1. Let $\ell := \lceil \mathrm{mc}/k_{\mathcal{C}} \rceil$. Send $(\mathsf{Init}, \mathsf{sid}, 2\ell)$ to $\mathcal{F}_{\mathsf{WVOLE}}$. $\mathcal{P}_{\mathrm{S}}$ obtains $\mathbf{U} \in \mathbb{F}_p^{2\ell \times k_{\mathcal{C}}}$, $\mathbf{V} \in \mathbb{F}_p^{2\ell \times n_{\mathcal{C}}}$ and $\overline{\mathbf{ch}} \in \mathcal{T}$ from $\mathcal{F}_{\mathsf{WVOLE}}$.
2. Set $\mathbf{V}_1 := \mathbf{V}_{[1..\ell]}, \mathbf{V}_2 := \mathbf{V}_{[\ell+1..2\ell]}$, $\mathbf{U}_1 := \mathbf{U}_{[1..\ell]}$ and $\mathbf{U}_2 := \mathbf{U}_{[\ell+1..2\ell]}$. We denote by $\mathbf{u}_1, \ldots, \mathbf{u}_\ell$ the $\ell$ rows of $\mathbf{U}_2$.
3. For each $j \in [\ell k_{\mathcal{C}}]$ add $\mathsf{cid}_j$ to its list $\mathtt{li}_{\mathrm{gen}}$.

$\mathcal{P}_{\mathrm{R}}$, upon obtaining $(\mathsf{Init}, \mathsf{sid}, 2\ell, \overline{\mathbf{ch}})$, for each $j \in [\ell k_{\mathcal{C}}]$ adds $\mathsf{cid}_j$ to $\mathtt{li}_{\mathrm{gen}}$.

**Commit** On input $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{cid}_{\iota_j}, m_j)_{j \in [k]})$ to $\mathcal{P}_{\mathrm{S}}$ where each $\mathsf{cid}_{\iota_j} \in \mathtt{li}_{\mathrm{gen}}$ and $m_j \in \mathbb{F}_p$:

1. $\mathcal{P}_{\mathrm{S}}$ for each $j \in [k]$ writes $x = \iota_j \bmod \ell$ and $y := (\iota_j - x)/\ell$. Then it broadcasts $d_j := m_j - \mathbf{U}[x+1, y+1]$.
2. $\mathcal{P}_{\mathrm{S}}$ for each $j \in [k]$ locally adds $(\mathsf{cid}_{\iota_j}, (m_j, f_{\iota_j}))$ to $\mathtt{mp}_{\mathrm{com}}$, while $\mathcal{P}_{\mathrm{R}}$ upon receiving the broadcast message adds $(\mathsf{cid}_{\iota_j}, (d_{\iota_j}, f_{\iota_j}))$ to $\mathtt{mp}_{\mathrm{com}}$.
3. Both parties remove each $\mathsf{cid}_{\iota_j}$ from their list $\mathtt{li}_{\mathrm{gen}}$.
4. Both parties send $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}} \| \mathtt{mp}_{\mathrm{com}}, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain $\mathbf{ch}$. Let $L = (\mathsf{cid}_j \mid (\mathsf{cid}_j, *) \in \mathtt{mp}_{\mathrm{com}})$. Then they add $(\overline{\mathbf{ch}} \| L, \mathbf{ch})$ to $\mathtt{li}_{\mathrm{chal}}$ and output $\mathbf{ch}$.

Fig. 10: Protocol $\Pi^{1-1}_{\mathsf{HOTCommit}}$ implementing $\mathcal{F}_{\mathsf{HOTCom}}$ for 1 receiver.

$\mathcal{P}_{\mathrm{S}}$ announces a correction value $d = x - \mathbf{U}_1[i, j]$ to be broadcast. $d$ allows $\mathcal{P}_{\mathrm{R}}$ to later correct its share of the level-1 VOLE commitment $(i, j)$ to the secret $x$ using the additive homomorphism of VOLE commitments.

*Hinting and opening.* To *hint* towards the receiver, the sender will simply reveal the committed value of the level-1 commitment but not the corresponding MAC in $\mathbf{U}_2$ (or the linear combination applied to these MACs). To *open* towards the receiver, $\mathcal{P}_{\mathrm{S}}$ first reveals all the MACs of the opened level-1 commitments to the receiver, which we denote for simplicity here as $A$. The sender then uses the random oracle on input $A$ and the transcript to generate the big-field VOLE challenge $\Delta_1$. It can compute all the level-1 VOLE commitment keys, namely the matrix $\mathbf{S} = \mathbf{U}_1 \Delta_1 + \mathbf{U}_2$ and give them to the receiver. Note that in a regular definition of VOLE, $\mathbf{S}$ is generated by the VOLE protocol and hidden from $\mathcal{P}_{\mathrm{S}}$.

$\Pi^{1-1}_{\mathsf{HOTCommit}}$ **(Part 2)**

**Hint** On input $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, i)$ to $\mathcal{P}_\mathrm{S}$, if $(\mathsf{cid}_j, (m_j, f_j)) \in \mathtt{mp}_{\mathrm{com}}$ for every $j \in [k]$ and if **Open** has not run yet, $\mathcal{P}_\mathrm{S}$ does the following:

**If** $i = 0$ broadcast $(\mathsf{cid}_j, m_j)_{j\in[k]}$ and add $(\mathsf{cid}_j, (m_j, f_j, \mathtt{pub}))$ to $\mathtt{mp}_{\mathrm{open}}$.

**If** $i = 1$

1. Sample $r_j \in \{0,1\}^\lambda$ and compute $g_j := f_j^{(0)} + f_j^{(1)}(\mathbf{u}_1, \ldots, \mathbf{u}_\ell)$.
2. For $j \in [k]$ send $(\mathsf{Query}, \mathsf{sid}, (m_j \| g_j \| r_j), \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ to obtain $c_j$.
3. Broadcast $(\mathsf{cid}_j, c_j)_{j\in[k]}$ and send $(\mathsf{cid}_j, m_j)_{j\in[k]}$ to $\mathcal{P}_\mathrm{R}$.
4. Add $(\mathsf{cid}_j, (m_j, g_j, r_j, c_j, f_j, \mathtt{priv}))$ to $\mathtt{mp}_{\mathrm{open}}$.

$\mathcal{P}_\mathrm{R}$, upon receiving $(\mathsf{cid}_j, m_j)_{j\in[k]}$ from $\mathcal{P}_\mathrm{S}$ via broadcast, if $(\mathsf{cid}_j, (d_j, f_j)) \in \mathtt{mp}_{\mathrm{com}}$ for every $j \in [k]$, if **Open** has not run yet and if no $(\mathsf{cid}_j, *) \in \mathtt{mp}_{\mathrm{open}}$, add each $(\mathsf{cid}_j, (m_j, d_j, f_j, \mathtt{pub}))$ to $\mathtt{mp}_{\mathrm{open}}$.

$\mathcal{P}_\mathrm{R}$, upon receiving $(\mathsf{cid}_j, m_j)_{j\in[k]}$ from $\mathcal{P}_\mathrm{S}$ and the broadcast $(\mathsf{cid}_j, c_j)_{j\in[k]}$, if $(\mathsf{cid}_j, (d_j, f_j)) \in \mathtt{mp}_{\mathrm{com}}$ for every $j \in [k]$, if **Open** has not run yet and if no $(\mathsf{cid}_j, *) \in \mathtt{mp}_{\mathrm{open}}$, add $(\mathsf{cid}_j, (m_j, c_j, d_j, f_j, \mathtt{priv}))$ to $\mathtt{mp}_{\mathrm{open}}$ for $j \in [k]$.

**Delayed Hint** On input $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$ to $\mathcal{P}_\mathrm{S}$, if $(\mathsf{cid}_j, (m_j, f_j)) \in \mathtt{mp}_{\mathrm{com}}$ for every $j \in [k]$ and if **Open** has not run yet, $\mathcal{P}_\mathrm{S}$ does the following:

1. Sample $r_j \in \{0,1\}^\lambda$ and compute $g_j := f_j^{(0)} + f_j^{(1)}(\mathbf{u}_1, \ldots, \mathbf{u}_\ell)$.
2. Send $(\mathsf{Query}, \mathsf{sid}, m_j \| g_j \| r_j, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain the response $c_j$ for each $j \in [k]$. Then broadcast $(\mathsf{cid}_j, c_j)_{j\in[k]}$.
3. Add $(\mathsf{cid}_j, (m_j, g_j, r_j, c_j, f_j, \mathtt{delay}))$ to $\mathtt{mp}_{\mathrm{open}}$.

$\mathcal{P}_\mathrm{R}$, upon receiving the broadcast $(\mathsf{cid}_j, c_j)_{j\in[k]}$, if $(\mathsf{cid}_j, (d_j, f_j)) \in \mathtt{mp}_{\mathrm{com}}$ for every $j \in [k]$, if **Open** has not run yet and if no $(\mathsf{cid}_j, *) \in \mathtt{mp}_{\mathrm{open}}$, adds $(\mathsf{cid}_j, (c_j, d_j, f_j, \mathtt{delay}))$ to $\mathtt{mp}_{\mathrm{open}}$ for each $j \in [k]$.

**Add** On input $(\mathsf{Add}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[3]})$ to $\mathcal{P}_\mathrm{S}$, if no call to **Open** was made yet, check if $(\mathsf{cid}_3, *) \notin \mathtt{mp}_{\mathrm{com}}$, $\mathsf{cid}_3 \notin \mathtt{li}_{\mathrm{gen}}$ and $\{(\mathsf{cid}_1, (A, f_1)), (\mathsf{cid}_2, (B, f_2))\} \subseteq \mathtt{mp}_{\mathrm{com}}$. If so:

- Each party adds $(\mathsf{cid}_3, (A + B, f_1 + f_2))$ to $\mathtt{mp}_{\mathrm{com}}$.

**Linear** On input $(\mathsf{Linear}, \mathsf{sid}, \mathsf{cid}_1, \mathsf{cid}_2, \alpha, \beta)$ to $\mathcal{P}_\mathrm{S}$, if **Open** was not called yet, check if $(\mathsf{cid}_2, *) \notin \mathtt{mp}_{\mathrm{com}}$, $\mathsf{cid}_2 \notin \mathtt{li}_{\mathrm{gen}}$ and $(\mathsf{cid}_1, (A, f)) \in \mathtt{mp}_{\mathrm{com}}$. If so:

- Set $g = \alpha f + \beta$
- Each party adds $(\mathsf{cid}_2, (\alpha A + \beta, g))$ to $\mathtt{mp}_{\mathrm{com}}$.

Fig. 11: Protocol $\Pi^{1-1}_{\mathsf{HOTCommit}}$ implementing $\mathcal{F}_{\mathsf{HOTCom}}$ for 1 receiver

The receiver, upon receiving $A$ and $\mathbf{S}$, first recomputes $\Delta_1$. With this information and the other messages from the sender, it can apply all the differences $d$ to the matrix $\mathbf{S}$, thus correcting all level-1 VOLE commitment keys of random messages to the actual messages. Then it applies all linear operations $f_j$ that were supposed to lead to the messages $m_j$ revealed earlier by the sender in *hint*. $\mathcal{P}_\mathrm{R}$ checks that the resulting keys are consistent with the opened MACs in $A$.

At this point, the commitment protocol is not secure: the sender could have lied when generating $\mathbf{S}$, which it generated *after* $\Delta_1$ is chosen, and the receiver does

$\Pi^{1-1}_{\mathsf{HOTCommit}}$ **(Part 3)**

**Open** On first input $(\mathsf{Open}, \mathsf{sid})$ $\mathcal{P}_{\mathrm{S}}$ does the following:

1. Compute $g_j := f_j^{(0)} + f_j^{(1)}(\mathbf{u}_1, \ldots, \mathbf{u}_\ell)$ and define

$$A_{\mathtt{pub}} := \{(\mathsf{cid}_j, m_j, g_j, f_j) \mid (\mathsf{cid}_j, (m_j, f_j)) \in \mathtt{mp}_{\mathrm{open}}\}$$
$$A_{\mathtt{priv}} := \{(\mathsf{cid}_j, c_j, f_j) \mid (\mathsf{cid}_j, (m_j, g_j, r_j, c_j, f_j, \mathtt{priv})) \in \mathtt{mp}_{\mathrm{open}}\}$$
$$A_{\mathtt{delay}} := \{(\mathsf{cid}_j, c_j, f_j) \mid (\mathsf{cid}_j, (m_j, g_j, r_j, c_j, f_j, \mathtt{delay})) \in \mathtt{mp}_{\mathrm{open}}\}$$
$$B_{\mathtt{priv}} := \{(\mathsf{cid}_j, m_j, g_j, r_j) \mid (\mathsf{cid}_j, (m_j, g_j, r_j, c_j, f_j, \mathtt{priv})) \in \mathtt{mp}_{\mathrm{open}}\}$$

2. $\mathcal{P}_{\mathrm{S}}$ sends $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}} \| A_{\mathtt{pub}} \| A_{\mathtt{priv}} \| A_{\mathtt{delay}}, p)$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtains $\Delta_1$.
3. $\mathcal{P}_{\mathrm{S}}$ computes $\mathbf{S} := \mathbf{U}_2 + \mathbf{U}_1\Delta_1$ and $\mathbf{D} := \mathbf{V}_2 + \mathbf{V}_1\Delta_1$.
4. $\mathcal{P}_{\mathrm{S}}$ broadcasts $(\mathbf{S}, \mathbf{D}, A_{\mathtt{pub}}, A_{\mathtt{delay}})$ and sends $B_{\mathtt{priv}}$ to $\mathcal{P}_{\mathrm{R}}$.
5. Let $m$ be the concatenation of all broadcast messages in order. Then send $(\mathsf{InitDone}, \mathsf{sid}, m)$ and $(\mathsf{Deliver}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{WVOLE}}$.

Upon receiving $B_{\mathtt{priv}}$ from $\mathcal{P}_{\mathrm{S}}$, $\mathcal{P}_{\mathrm{R}}$ does the following:

1. Obtain $(\mathsf{Done}, \mathsf{sid}, m, \Delta_0)$ from $\mathcal{F}_{\mathsf{WVOLE}}$.
2. Check that $\mathbf{S}, \mathbf{D}, A_{\mathtt{pub}}, A_{\mathtt{delay}}$ as contained in $m$ was broadcast on $\mathcal{F}_{\mathsf{GBB}}$. If not, then abort.
3. For each $(\mathsf{cid}_j, m_j, g_j, f_j) \in A_{\mathtt{pub}}$ if $(\mathsf{cid}_j, (m_j, d_j, f_j, \mathtt{pub})) \in \mathtt{li}_{\mathrm{open}}$ for every $j \in [k]$ replace each $(\mathsf{cid}_j, (m_j, d_j, f_j, \mathtt{pub}))$ in $\mathtt{mp}_{\mathrm{open}}$ with $(\mathsf{cid}_j, (m_j, g_j, d_j, f_j, \mathtt{pub}))$.
4. For each $(\mathsf{cid}_j, m_j, g_j, r_j) \in B_{\mathtt{priv}}$, if $(\mathsf{cid}_j, (m_j, c_j, d_j, f_j, \mathtt{priv})) \in \mathtt{mp}_{\mathrm{open}}$ for every $j \in [k]$ send $(\mathsf{Query}, \mathsf{sid}, m_j \| g_j \| r_j, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and check if the response is $c_j$. If so, then replace each $(\mathsf{cid}_j, (m_j, c_j, d_j, f_j, \mathtt{priv}))$ in $\mathtt{mp}_{\mathrm{open}}$ with $(\mathsf{cid}_j, (m_j, g_j, c_j, d_j, f_j, \mathtt{priv}))$. Otherwise abort.
5. Obtain $(\mathsf{Corr}, \mathsf{sid}, \mathbf{Q})$ from $\mathcal{F}_{\mathsf{WVOLE}}$ where $\mathbf{Q} \in \mathbb{F}_p^{2\ell \times n_{\mathcal{C}}}$. Set $\mathbf{Q}_1 := \mathbf{Q}_{[1..\ell]}$ and $\mathbf{Q}_2 := \mathbf{Q}_{[\ell+1..2\ell]}$.

**Delayed Open** On second input $(\mathsf{Open}, \mathsf{sid})$ $\mathcal{P}_{\mathrm{S}}$ computes and broadcasts

$$A_{\mathtt{delay,open}} := \{(\mathsf{cid}_j, m_j, g_j, r_j, f_j) \mid (\mathsf{cid}_j, (m_j, g_j, r_j, c_j, f_j, \mathtt{delay})) \in \mathtt{mp}_{\mathrm{open}}\}$$

Upon observing $A_{\mathtt{delay,open}}$ on $\mathcal{F}_{\mathsf{GBB}}$ broadcast by $\mathcal{P}_{\mathrm{S}}$, $\mathcal{P}_{\mathrm{R}}$ does the following:

1. For each $(\mathsf{cid}_j, m_j, g_j, r_j, f_j) \in A_{\mathtt{delay,open}}$, if $(\mathsf{cid}_j, (c_j, d_j, f_j, \mathtt{delay})) \in \mathtt{mp}_{\mathrm{open}}$ for every $j \in [k]$ and $(\mathsf{cid}_j, c_j, f_j) \in A_{\mathtt{delay}}$ send $(\mathsf{Query}, \mathsf{sid}, m_j \| g_j \| r_j, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and check if the response is $c_j$.
2. If $c_j$ is correct then replace each $(\mathsf{cid}_j, (c_j, d_j, f_j, \mathtt{delay}))$ in $\mathtt{mp}_{\mathrm{open}}$ with $(\mathsf{cid}_j, (m_j, g_j, r_j, d_j, f_j, \mathtt{delay}))$. Otherwise abort.

**Challenges** Upon input $(\mathsf{Challenges}, \mathsf{sid})$ to any party, send $(\mathsf{Challenge}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{WVOLE}}$. If it returns $\perp$ return $\perp$, otherwise obtain $\overline{\mathbf{ch}}$. Then go through every **Commit** broadcast on $\mathcal{F}_{\mathsf{GBB}}$, beginning with the first posted, and compute $\mathtt{li}_{\mathrm{chal}}$ as $\mathcal{P}_{\mathrm{R}}$ does in **Commit** using $\overline{\mathbf{ch}}$. Then output $\mathtt{li}_{\mathrm{chal}}$.

Fig. 12: Protocol $\Pi^{1-1}_{\mathsf{HOTCommit}}$ implementing $\mathcal{F}_{\mathsf{HOTCom}}$ for 1 receiver

$\Pi^{1-1}_{\mathsf{HOTCommit}}$ **(Part 4)**

**Verify** On input $(\mathsf{Verify}, \mathsf{sid}, (\mathsf{cid}'_j, m'_j)_{j\in[k]})$ to $\mathcal{P}_{\mathrm{R}}$ where all $(\mathsf{cid}'_j, \dots) \in \mathtt{mp}_{\mathrm{open}}$, if **Open** was finished and $\mathcal{P}_{\mathrm{R}}$ did not abort it does the following:

1. Compute

$$A_{\mathtt{pub}} := \{(\mathsf{cid}_j, m_j, g_j, f_j) \mid (\mathsf{cid}_j, (m_j, g_j, d_j, f_j, \mathtt{pub})) \in \mathtt{mp}_{\mathrm{open}}\}$$
$$A_{\mathtt{priv}} := \{(\mathsf{cid}_j, c_j, f_j) \mid (\mathsf{cid}_j, (m_j, g_j, c_j, d_j, f_j, \mathtt{priv})) \in \mathtt{mp}_{\mathrm{open}}\}$$
$$A_{\mathtt{delay}} := \left\{(\mathsf{cid}_j, c_j, f_j) \mid \begin{matrix}(\mathsf{cid}_j, (m_j, g_j, r_j, d_j, f_j, \mathtt{delay})) \in \mathtt{mp}_{\mathrm{open}} \text{ or} \\ (\mathsf{cid}_j, (c_j, d_j, f_j, \mathtt{delay})) \in \mathtt{mp}_{\mathrm{open}}\end{matrix}\right\}$$

2. Send $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}} \| A_{\mathtt{pub}} \| A_{\mathtt{priv}} \| A_{\mathtt{delay}}, p)$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain $\Delta_1$.
3. Let $\mathbf{\Gamma} \in \mathbb{F}_p^{\ell \times k_C}$ be such that

$$\mathbf{\Gamma}[i,j] := \begin{cases} d_{(j-1)\ell+i-1} & \text{if } (\mathsf{cid}'_j, (m'_j, g_j, d_j, f_j, \mathtt{pub})) \in \mathtt{mp}_{\mathrm{open}} \\ & \text{or } (\mathsf{cid}'_j, (m'_j, g_j, c_j, d_j, f_j, \mathtt{priv})) \in \mathtt{mp}_{\mathrm{open}} \\ & \text{or } (\mathsf{cid}'_j, (m'_j, g_j, r_j, d_j, f_j, \mathtt{delay})) \in \mathtt{mp}_{\mathrm{open}} \\ 0 & \text{otherwise} \end{cases}$$

4. Let $f_j$ be all the polynomials used to define $\mathbf{\Gamma}$.
5. Set $\mathbf{S}' := \mathbf{S} + \mathbf{\Gamma}\Delta_1$ and denote row $i$ as $\mathbf{s}'_j$
6. For each $(\mathsf{cid}'_j, (m'_j, g_j, d_j, f_j, \mathtt{pub}))$, $(\mathsf{cid}'_j, (m'_j, g_j, c_j, d_j, f_j, \mathtt{priv}))$ and $(\mathsf{cid}'_j, (m'_j, g_j, r_j, d_j, f_j, \mathtt{delay}))$ in $\mathtt{mp}_{\mathrm{open}}$ check that

$$g_j = f_j^{(0)} + f_j^{(1)}(\mathbf{s}'_1, \dots, \mathbf{s}'_\ell) - m_j \Delta_1$$

7. Check that $\mathbf{D} = \mathbf{Q}_2 + \mathbf{Q}_1\Delta_1 - \mathcal{C}(\mathbf{S}) \cdot \mathsf{diag}(\Delta_0)$

**Audit** On input $(\mathsf{Audit}, \mathsf{sid}, (\mathsf{cid}'_j, f'_j)_{j\in[k]})$ $\mathcal{P}_{\mathrm{A}}$ does the following:

1. If $\mathcal{P}_{\mathrm{S}}$ or $\mathcal{P}_{\mathrm{R}}$ broadcast $(\mathsf{Abort}, \mathsf{sid})$ then return $(\mathsf{AuditFailed}, \mathsf{sid})$.
2. Send $(\mathsf{Audit}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{WVOLE}}$. If it returns $(\mathsf{AuditFailed}, \mathsf{sid})$ return $(\mathsf{AuditFailed}, \mathsf{sid})$, otherwise store $\ell, \overline{\mathbf{ch}}, m, \Delta_0, \mathbf{Q}$.
3. If $m$ is not identical with the messages broadcast by $\mathcal{P}_{\mathrm{S}}$ in the respective order then return $(\mathsf{AuditFailed}, \mathsf{sid})$.
4. Set $A_{\mathtt{priv}} := \emptyset$ and recompute $\mathtt{li}_{\mathrm{gen}}$ like $\mathcal{P}_{\mathrm{R}}$ in **Init**.
5. For each **Commit**, **Hint**, **Delayed Hint**, **Add**, **Linear** perform the actions that $\mathcal{P}_{\mathrm{R}}$ performs.
6. For each **Hint** with $i = 1$ and a broadcast message $(\mathsf{cid}_j, c_j)_{j\in[k]}$ if $(\mathsf{cid}_j, (d_j, f_j)) \in \mathtt{mp}_{\mathrm{com}}$ and if no $(\mathsf{cid}_j, \dots) \in \mathtt{li}_{\mathrm{open}}$ add all $(\mathsf{cid}_j, c_j, f_j)$ to $A_{\mathtt{priv}}$.
7. Run steps 2 and 3 of $\mathcal{P}_{\mathrm{R}}$ in **Open** and all steps in **Delayed Open**.
8. Recompute $A_{\mathtt{pub}}, A_{\mathtt{delay}}$ like $\mathcal{P}_{\mathrm{R}}$ does during **Verify**. If $(\mathsf{cid}'_j, m'_j, \dots, f'_j) \notin A_{\mathtt{pub}} \cup A_{\mathtt{delay}}$ for some $\mathsf{cid}'_j$ return $(\mathsf{AuditFailed}, \mathsf{sid})$.
9. Run **Verify** with input $(\mathsf{cid}'_j, m'_j)_{j\in[k]}$ and $A_{\mathtt{priv}}$ as computed above. Output $(\mathsf{Audit}, \mathsf{sid}, (\mathsf{cid}'_j, m'_j)_{j\in[k]})$ if it accepts, otherwise output $(\mathsf{AuditFailed}, \mathsf{sid})$.

Fig. 13: Protocol $\Pi^{1-1}_{\mathsf{HOTCommit}}$ implementing $\mathcal{F}_{\mathsf{HOTCom}}$ for 1 receiver

not have any guarantee yet that $\mathbf{S}$ is well-formed. By changing $\mathbf{S}$ the sender could have forged the level-1 VOLE keys of the receiver, easily allowing it to pass the MAC check outlined above. We therefore now explain how the receiver uses $\mathbf{Q}$ and $\Delta_0$ generated by $\mathcal{F}_{\mathsf{WVOLE}}$ to verify that $\mathbf{S}$ was formed correctly.

*Checking the VOLE key matrix* $\mathbf{S}$. The receiver can check the correctness of $\mathbf{S}$ based on its share of the correlation that $\mathcal{F}_{\mathsf{WVOLE}}$ outputs. We call the correlation formed by $\mathcal{F}_{\mathsf{WVOLE}}$ the "level-0" VOLEs. Recall that the receiver obtains $\mathbf{Q} = \mathbf{UG}\mathsf{diag}(\Delta_0)+\mathbf{V}$, by definition of $\mathcal{F}_{\mathsf{WVOLE}}$. This is a subspace-VOLE correlation, where all of $\mathbf{U}$ is now the committed secret: the secrets of the initial random level-1 VOLEs ($\mathbf{U}_1$) are identical to the first $\ell$ rows of the level-0 VOLE correlation. Moreover, the MACs ($\mathbf{U}_2$) of the level-1 VOLEs are identical to the lower half of the level-0 VOLE correlation. Thus, they both form the secret of the level-0 VOLE, and $\mathbf{S}$ is a publicly known linear transform of the rows of the secret of the level-0 VOLEs, i.e. of $\mathbf{U}$. Since this is a row-wise linear transform on $\mathbf{U}$, we can check correctness of $\mathbf{S}$ using the level-0 VOLE correlation.

Towards this, the sender additionally sends $\mathbf{D} = \mathbf{V}_2 + \mathbf{V}_1\Delta_1$ alongside $\mathbf{S}$. This applies the same linear transform which generated $\mathbf{S}$ from $\mathbf{U}$, but this time on the MACs of the level-0 VOLEs (i.e. on $\mathbf{V}$). Then, the receiver obtains $(\mathsf{Done}, \mathsf{sid}, m, \Delta_0)$ and $(\mathsf{Corr}, \mathsf{sid}, \mathbf{Q})$ from $\mathcal{F}_{\mathsf{WVOLE}}$. Denoting the first $\ell$ rows of $\mathbf{Q}$ as $\mathbf{Q}_1$ and lower $\ell$ rows as $\mathbf{Q}_2$, the receiver can compute $\mathbf{Q}_2 + \mathbf{Q}_1\Delta_1$. If $\mathbf{S}, \mathbf{D}$ were formed correctly, then it must hold that

$$\underbrace{\mathbf{Q}_2 + \mathbf{Q}_1\Delta_1}_{\text{level-0 VOLE Key}} = \underbrace{\mathbf{V}_2 + \mathbf{V}_1\Delta_1}_{\text{level-0 VOLE MAC } \mathbf{D}} + \underbrace{\mathcal{C}(\mathbf{S})}_{\text{level-0 VOLE message}} \cdot \mathsf{diag}(\Delta_0)$$

This equation is checked by the receiver, which broadcasts $(\mathsf{Abort}, \mathsf{sid})$ if the check fails. Security follows because $\mathbf{S}$ is encoded in the linear code $\mathcal{C}$ using the public generator matrix $\mathbf{G}$ during the check, while $\mathbf{Q}$ of $\mathcal{F}_{\mathsf{WVOLE}}$ is also correlated using $\mathcal{C}$. To pass the check with an inconsistent input, one can show that $\mathcal{P}_{\mathrm{S}}$ must essentially guess $\Delta_0$ in at least $d_{\mathcal{C}}$ positions before it is chosen.

*Complications due to delayed and private hints.* While this description describes the general commitment protocol that we construct, it must also perform **Hint** with a private opening and **Delayed Hint** as specified in $\mathcal{F}_{\mathsf{HOTCom}}$. While implementing this complicates the protocol description on a formal level, the ideas for achieving these properties are simple: For any **Hint** with a private opening the sender hashes the message $m$, together with some randomness $r$, into a commitment $c$ using the random oracle $\mathcal{F}_{\mathsf{GpRO}}$. $c$ is broadcast, while $m, r$ are sent to the receiver. Later, $c$ is part of the challenge to derive $\Delta_1$ (thus binding $\mathcal{P}_{\mathrm{S}}$ to $m$) and, upon receiving the MAC in private, the receiver can check the MAC and validity of the opening of the commitment $c$. This also allows third parties to recompute $\Delta_1$ as only $c$ is used to derive it, which is crucial for auditing. **Delayed Hint** works similarly: there, the message $m$ whose opening is delayed will be committed to in a commitment $c$ that is broadcast. Then, when the delayed hint is opened during **Delayed Open**, the message and randomness (as well as

the MAC) are revealed. The receiver can again check if the commitment indeed contains $m$ and that the MAC is valid. This time, however, $m$ as well as the MAC will be broadcast, as they are not private outputs.

*Audits and Challenges.* To perform **Audit**, the auditor will collect all messages to be verified from the broadcast channel. It then uses $\mathcal{F}_{\mathsf{WVOLE}}$'s **Audit** to obtain the same $\mathbf{Q}$ matrix of the level-0 VOLE correlation that $\mathcal{P}_{\mathrm{R}}$ obtained (as well as $\Delta_0$) and then basically performs the same operations and checks that $\mathcal{P}_{\mathrm{R}}$ did during the protocol to apply linear homomorphisms and check openings. One crucial difference is that $\mathcal{P}_{\mathrm{A}}$ does not have access to *private* openings done via **Private Hint**, but it has access to the commitments that contain the messages (which are broadcast). This is sufficient to recompute $\Delta_1$.

To ensure that $\Delta_1$ is unpredictable to the adversary, the computation involves the use of the challenge $\overline{\mathbf{ch}} \in \{0,1\}^{2\lambda}$ generated by the hybrid functionality. This is only generated after the inputs of the sender to $\mathcal{F}_{\mathsf{WVOLE}}$ are fixed. By construction, we ensure that the challenge $\Delta_0$ of the level-0 VOLE correlation is only chosen after all other messages (and also $\Delta_1$) are fixed. Figures 10, 11, 12 and 13 contain the full protocol.

**Theorem 1.** *Let $\lambda$ be the security parameter, $\mathcal{C}$ be an $[n_{\mathcal{C}}, k_{\mathcal{C}}, d_{\mathcal{C}}]$-linear code over $\mathbb{F}_p$ and $N < p$. The protocol $\Pi^{1-1}_{\mathsf{HOTCommit}}$ securely implements $\mathcal{F}_{\mathsf{HOTCom}}$ against any PPT static adversary $\mathcal{A}$ corrupting at most both $\mathcal{P}_S, \mathcal{P}_R$ but never $\mathcal{P}_A$, in the $\mathcal{F}^{p,N,\mathcal{C}}_{\mathsf{WVOLE}}$-hybrid model with access to global subroutines $\mathcal{F}_{\mathsf{GBB}}, \mathcal{F}_{\mathsf{GpRO}}$. $\mathcal{A}$ makes at most $Q$ queries to the Random Oracle. The distinguishing advantage of $\mathcal{Z}$ is at most $Q/2^{\lambda} + Q/N^{d_C} + Q/p + Q^2/2^{2\lambda} + Q^2/p$, and we use $\mathcal{T} = 2^{2\lambda}$ as challenge space for $\mathcal{F}_{\mathsf{WVOLE}}$.*

The proof can be found in Appendix E. The $Q^2/p$-term in the theorem is only an artifact of the simplified construction of the protocol: it comes from random oracle calls on inputs from $\mathbb{F}_p$ that have to generate a challenge. In practice, this can be resolved by always requiring 2-3 values to be committed simultaneously.

The proof, while lengthy, is relatively straightforward: we provide two simulation strategies, depending on the sender being corrupted or not. If the sender is honest, then messages and commitments send during **Commit**, **Hint** or **Delayed Hint** are essentially random. During **Open**, we carefully choose the $g_j$ to fit based on the simulated messages and outputs of the ideal functionality. If the sender is dishonest, then we use the extraction strategy from [7], which works because the challenges $\Delta_1, \Delta_0$ are unpredictable to the sender and because the transcript influences $\Delta_0$ generated by $\mathcal{F}_{\mathsf{WVOLE}}$ as well. We use **Challenge Test** of $\mathcal{F}_{\mathsf{WVOLE}}$ to simulate random oracle responses for choices of $\mathbf{U}, \mathbf{V}$.

### 4.3 From single-receiver to multi-receiver one-time commitments

We now extend the previous protocol to the multi-receiver setting where $|\mathcal{P}_{\mathrm{R}}| > 1$ using the protocol $\Pi^{1-n}_{\mathsf{HOTCommit}}$. It is realized in the $\mathcal{F}_{\mathsf{GBB}}, \mathcal{F}_{\mathsf{GpRO}}, \mathcal{F}_{\mathsf{HOTCom}}$-hybrid

model and described in Figure 20 and Figure 21 in Appendix F. $\Pi^{1-n}_{\mathsf{HOTCommit}}$ runs an instance of $\mathcal{F}_{\mathsf{HOTCom}}$ with every receiver, committing to the same value everywhere. To show consistency, it uses the linearly homomorphic properties of the commitment scheme.

In more detail, the sender $\mathcal{P}_{\mathrm{S}}$ initializes one instance of $\mathcal{F}_{\mathsf{HOTCom}}$ with each party $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$, which we denote as $\mathcal{F}^i_{\mathsf{HOTCom}}$. During **Commit, Hint, Delayed Hint, Open, Add, Linear** and **Verify** $\mathcal{P}_{\mathrm{S}}$ will send the same message to each $\mathcal{F}^i_{\mathsf{HOTCom}}$. For **Hint** with a private receiver the sender only has to send the message to $\mathcal{F}^i_{\mathsf{HOTCom}}$ with receiver $\mathcal{P}_i$. Every message in **Delayed Hint** will be revealed during **Delayed Open**.

During **Commit** the sender could commit to different messages between two or more instances of $\mathcal{F}^i_{\mathsf{HOTCom}}$. This is easy to catch for all messages that are opened through **Hint** or **Delayed Open** as $\mathcal{P}_{\mathrm{S}}$ will send the correct value to $\mathcal{F}_{\mathsf{GBB}}$ before revealing it at each $\mathcal{F}^i_{\mathsf{HOTCom}}$. Each $\mathcal{P}_i$ can then compare the value they received from $\mathcal{F}^i_{\mathsf{HOTCom}}$ with the value that was sent to $\mathcal{F}_{\mathsf{GBB}}$. To further ensure that the values that are sent via **Private Hint** are consistent, the parties perform a **Consistency Check** procedure during **Open**: there, they sample a universal hash function which is applied to all committed values. Afterwards, they open the outcome of the universal hash. If any difference is observed, then the commitments are inconsistent.

To prevent leakage of secrets, **Consistency Check** uses an additional value $r$ to which the sender committed herself to during **Init**. In order to sample a universal hash function, the parties use the challenge value from the most recent **Commit** to derive as many challenge values as there are commitment values. Later, any auditor $\mathcal{P}_{\mathrm{A}}$ can verify the correctness of any commitment opening by inspecting $\mathcal{F}_{\mathsf{GBB}}$ and verifying outputs with each $\mathcal{F}^i_{\mathsf{HOTCom}}$. To implement **Challenges**, note that each $\mathcal{F}^i_{\mathsf{HOTCom}}$ outputs a challenge upon commitment to the sender and receiver, but makes the value also accessible to any other party. Every party $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$ obtains the challenges of all functionalities and hashes these together using $\mathcal{F}_{\mathsf{GpRO}}$. In Appendix F we prove the following theorem:

**Theorem 2.** *In the* $(\mathcal{F}_{\mathsf{GBB}}, \mathcal{F}_{\mathsf{GpRO}}, \mathcal{F}^{1-1}_{\mathsf{HOTCom}})$*-hybrid model the protocol* $\Pi^{1-n}_{\mathsf{HOTCommit}}$ *implements* $\mathcal{F}_{\mathsf{HOTCom}}$ *over the field* $\mathbb{F}_p$ *with statistical security against any static active adversary corrupting at most all of* $\mathcal{P}_S, \mathcal{P}_R$ *but never* $\mathcal{P}_A$*. The adversary makes at most* $Q$ *queries to* $\mathcal{F}_{\mathsf{GpRO}}$*, and the distinguishing advantage at most* $Q/p + Q/2^\lambda + Q^2/2^{2\lambda}$*. The challenge space is* $\mathcal{T} = \{0,1\}^{2\lambda}$*.*

The proof strategy again distinguishes between an honest and dishonest sender. If the sender is honest, then we simply program the $\mathcal{F}^i_{\mathsf{HOTCom}}$ functionalities to make the correct outputs and use the hiding property of the universal hash to simulate the consistency check. If the sender is corrupted, then the adversary can cheat by either committing to different values to each party or by sending hints that differ from the commitments. While the latter is easy to catch, the former is caught through the consistency check which uses that the challenge for

the check is unpredictable by using a challenge value generated by $\mathcal{F}_{\mathsf{HOTCom}}$. To make this consistent, we emulate random oracle responses for challenge queries by outputting what $\mathcal{F}_{\mathsf{HOTCom}}$ outputs. This is guaranteed to work as the simulator controls challenge generation of the $\mathcal{F}^{i}_{\mathsf{HOTCom}}$ instances, whose challenges are unpredictable to the attacker before actually committing to a value (thus allowing to program). This simulation also succeeds if all receivers are corrupt.

## 5 Offline Phase

During the online phase of the MPC protocol, pre-generated random values and triples are used to evaluate the circuit. Therefore, we now introduce the functionality $\mathcal{F}_{\mathsf{Offline}}$ in Figure 14 that generates these with UC security. Note that the triples generated by the functionality might not be well-formed. However, this will be corrected for during the online phase.

We now give a short description of $\mathcal{F}_{\mathsf{Offline}}$. Upon initialization in **Initialize**, the parties have to decide how many sharings for random values and triples can be generated later, which is correlated to the number of input and multiplication gates of the circuit $\mathcal{C}$ that is evaluated by a protocol using $\mathcal{F}_{\mathsf{Offline}}$. For each of these challenges, the offline functionality will reserve certain IDs which refer to them internally. These IDs can be used by the parties to *reliably* open the sharings later. These are either triple IDs ($\mathsf{tid}$) or IDs for input sharings ($\mathsf{rid}$). During **Input** and **Triples** random additive sharings will be generated among all parties. The adversary is allowed to pick his shares and make triples incorrect using additive terms. Moreover, sharings that exist can have linear transforms applied to them using commands in the functionality. However, the ID of the result must always differ from IDs reserved for inputs and triples for technical reasons. The functionality stores the shares internally in the map $\mathtt{mp}_{\mathrm{val}}$.

Finally, parties are allowed to open their sharings either privately towards a party ($\mathtt{type} = 0$) or in public ($\mathtt{type} = 1$) using **Open**. In either case, an attacker is allowed to add an additive error to the revealed value, but this cheating is noted by the functionality. Parties can also decide to delay an opening to a later stage ($\mathtt{type} = 2$), and release such a delayed sharing later on. This is similar to the **Open** and **Delayed Open** of $\mathcal{F}_{\mathsf{HOTCom}}$, but where now all parties can be senders. Opened values are stored internally in the list $\mathtt{li}_{\mathrm{open}}$, together with attributes if they are private or delayed openings.

At this point, the opened values might be wrong and cannot yet be verified. **Release** upon first call will make all opened values verifiable. Thereafter, a call to **Full Release** makes openings also verifiable to the auditor. Similar to $\mathcal{F}_{\mathsf{HOTCom}}$ the functionality permits opening values in private, which cannot be verified by the auditor. The state of verifiability is stored in the flag $\mathtt{fl}_{\mathrm{ver}}$.

To ease use with other protocols in the Fiat-Shamir setting, the functionality generates challenge values upon committing by the participants and an adversary can test which challenges its controlled parties' inputs would generate before

committing using **Triples**. We keep track of the number of generated challenges using *ccount*. If not all parties are corrupted, the attacker must choose a challenge consistent with the remaining state which we enforce using an extra *min* variable. To allow simulation of the protocol in the head of the attacker if all parties are corrupted, such an attacker can also *reset* the functionality at any point. This resets *ccount* as well, allowing it to also set the challenge to a value related to previous states. The same challenge values can later be queried by the auditor. As before, we use $\mathtt{mp}_{\text{chal}}$ to keep track of all generated challenges and store the ultimately chosen challenge in the flag $\mathtt{fl}_{\text{ch}}$.

### 5.1 Offline Protocol

The offline phase is implemented through $\Pi_{\mathsf{Offline}}$ shown in Figure 16 Figure 17. The protocol generates authenticated random values and pairwise authenticated partial triples, in a style similar to [30]. Just as in the functionality, each secret in the protocol is additively shared. Each party $\mathcal{P}_i$ is a sender in an instance of $\mathcal{F}_{\mathsf{HOTCom}}$ where it is committed to for its share and where the functionality has $n$ receivers. The shares can be opened and verified via $\mathcal{F}_{\mathsf{HOTCom}}$, and linear operations can be applied as well. Finally, the auditor $\mathcal{P}_{\text{A}}$ can verify outputs by using the **Audit** interfaces of all $\mathcal{F}_{\mathsf{HOTCom}}$ instances to obtain the shares and comparing with its input.

However, one difficulty is that $\mathcal{F}_{\mathsf{Offline}}$ must output potentially faulty triples $(a^i, b^i, c^i)$ such that, if no party cheats, $\sum a^i \cdot \sum b^i = \sum c^i$. We achieve this by using an instance of $\mathcal{F}_{\mathsf{OLE}}$ between each ordered pair of parties $\mathcal{P}_i, \mathcal{P}_j$. Here, we follow [30] and many earlier works that each $c^i$ can be computed as

$$c^i = a^i \cdot b^i + \sum\nolimits_{j<i} a^j \cdot b^i + \sum\nolimits_{j<i} a^i \cdot b^j$$

While $a^i \cdot b^i$ can be computed by each $\mathcal{P}_i$ locally, it uses $\mathcal{F}_{\mathsf{OLE}}$ together with every $\mathcal{P}_j$ to obtain additive shares of $a^j \cdot b^i$, $a^i \cdot b^j$ which it adds to $a^i \cdot b^i$ before committing to the sum. In Appendix G we prove the following statement:

**Theorem 3.** *In the* $(\mathcal{F}_{\mathsf{GBB}}, \mathcal{F}_{\mathsf{HOTCom}}, \mathcal{F}_{\mathsf{OLE}})$*-hybrid model, the protocol* $\Pi_{\mathsf{Offline}}$ *securely implements* $\mathcal{F}_{\mathsf{Offline}}$ *with security against any static active adversary corrupting at most all parties in* $\mathcal{P}$ *but never* $\mathcal{P}_A$*. If the adversary makes at most* $Q$ *queries to* $\mathcal{F}_{\mathsf{GpRO}}$ *then the distinguishing advantage is at most* $Q^2/2^{2\lambda}$*. The provided challenge space is* $\mathcal{T} = \mathbb{F}_p^*$*.*

To prove the statement, we provide different simulators depending on there being an honest party (except $\mathcal{P}_{\text{A}}$) or if all parties are corrupted. If an honest party exists, then the simulation strategy mainly follows [30] with small adjustments for the commitments. If all parties are corrupted, we alter the strategy to generate unpredictable challenges for the attacker based on $\mathcal{F}_{\mathsf{GpRO}}$ queries. Here, we can emulate consistent $\mathcal{F}_{\mathsf{GpRO}}$ outputs by using that its inputs, combined with the inputs to all $\mathcal{F}^i_{\mathsf{HOTCom}}$, fix all inputs to the $\mathcal{F}_{\mathsf{Offline}}$ functionality necessary to compute the challenge. We then use $\mathsf{reset}$ until the adversary finishes a transcript to obtain more challenges if needed for the simulation.

$\mathcal{F}_{\mathsf{Offline}}$ **(Part 1)**

$\mathcal{F}_{\mathsf{Offline}}$ interacts with a set of parties $\mathcal{P}$, an auditor $\mathcal{P}_{\mathrm{A}}$ and an adversary $\mathcal{A}$. $\mathcal{A}$ may corrupt a subset of parties $\tilde{\mathcal{P}} \subseteq \mathcal{P}$ but never $\mathcal{P}_{\mathrm{A}}$. Let $\mathbb{F}$ be a field. The adversary or any party may send $(\mathsf{Abort}, \mathsf{sid})$ at any point, whereupon $\mathcal{F}_{\mathsf{Offline}}$ sends $(\mathsf{Abort}, \mathsf{sid})$ to all parties and aborts. The provided challenge space is $\mathcal{T} = \mathbb{F}_p^*$. Let $\mathtt{fl}_{\mathrm{ch}}, \mathtt{fl}_{\mathrm{ver}}$ be flags that are initially $\perp$, $\mathtt{li}_{\mathrm{open}}$ be an initially empty list and $\mathtt{mp}_{\mathrm{val}}, \mathtt{mp}_{\mathrm{chal}}$ be empty maps. If all parties are corrupt and $\mathcal{A}$ sends $\mathsf{reset}$, reset $\mathtt{mp}_{\mathrm{val}}$ and $\mathtt{fl}_{\mathrm{ch}}$.

**Initialize** On first input $(\mathsf{Init}, \mathsf{sid}, \mathsf{in}, \mathsf{m})$ from all parties in $\mathcal{P}$:

1. Store $\mathsf{in}$ for the number of random values and $\mathsf{m}$ for the triples.
2. Set $\mathsf{tid} = 1$, $\mathsf{rid} = 1$, $ccount = 0$, $\mathtt{fl}_{\mathrm{ver}} = \perp$ and $\mathtt{fl}_{\mathrm{ch}} = \perp$.

**Input** On first input $(\mathsf{Input}, \mathsf{sid})$ from all parties parties in $\mathcal{P}$ and if $\mathtt{fl}_{\mathrm{ver}} = \perp$:

1. For each $l \in [\mathsf{in}]$ and all honest parties $\mathcal{P}_i \in \mathcal{P} \setminus \tilde{\mathcal{P}}$ choose $(r_l^i)_{l \in [\mathsf{in}]} \xleftarrow{\$} \mathbb{F}$.
2. Wait for $\mathcal{A}$ to send $(\mathsf{rid}_l, r_l^i)_{l \in [\mathsf{in}]}$ for every $\mathcal{P}_i \in \tilde{\mathcal{P}}$.
3. Set $r_l = \sum r_l^i$ and store $(\mathsf{rid}_l, r_i)_{l \in [\mathsf{in}]}$ in $\mathtt{mp}_{\mathrm{val}}$. Send $(\mathsf{ShareInput}, \mathsf{sid}, (\mathsf{rid}, r^i))_{l \in [\mathsf{in}]}$ to each $\mathcal{P}_i$.

**Triples** On first input $(\mathsf{Triples}, \mathsf{sid})$ from all parties in $\mathcal{P}$ and if $\mathtt{fl}_{\mathrm{ver}} = \perp$:

1. For all honest parties $\mathcal{P}_i \in \mathcal{P} \setminus \tilde{\mathcal{P}}$ sample $(a_l^i, b_l^i)_{l \in [\mathsf{m}]} \xleftarrow{\$} \mathbb{F}$. Increase $ccount$ and set $min = ccount$.
2. If $|\tilde{P}| = 0$
   (a) Set $a = \sum_{i \in \mathcal{P}} a_l^i$, $b = \sum_{i \in \mathcal{P}} b_l^i$ and sample $c_l^i \xleftarrow{\$} \mathbb{F}$ subject to the constraint that $c_l = \sum_{i \in \mathcal{P}} c_l^i$.
   (b) Append $(\mathsf{tid}_a, a), (\mathsf{tid}_b, b)_l, (\mathsf{tid}_c, c)$ to $\mathtt{mp}_{\mathrm{val}}$ and sample $\mathtt{fl}_{\mathrm{ch}} \xleftarrow{\$} \mathcal{T}$.
3. If $|\tilde{P}| > 0$
   (a) Wait for $\mathcal{A}$ to send $(\mathsf{Values}, \mathsf{sid}, ((a^i, b^i, c^i)_{i \in \tilde{\mathcal{P}}}, \Delta_c, (\delta_a^j, \delta_b^j)_{j \in \mathcal{P} \setminus \tilde{\mathcal{P}}})))$ and let $\mathsf{state}$ be a copy of $\mathtt{mp}_{\mathrm{val}}$.
   (b) Set $a = \sum_{i \in \mathcal{P}} a^i$, $b = \sum_{i \in \mathcal{P}} b^i$ and $c = a \cdot b + \sum_{i \in \mathcal{P} \setminus \tilde{\mathcal{P}}} a^i \cdot \delta_a^i + b^i \cdot \delta_b^i + \Delta_c$.
   (c) For all honest parties $\mathcal{P}_i \in \mathcal{P} \setminus \tilde{\mathcal{P}}$ sample $c^i \xleftarrow{\$} \mathbb{F}$ subject to the constraint that $c = \sum_{i \in \mathcal{P}} c^i$.
   (d) Append $(\mathsf{tid}_a, a), (\mathsf{tid}_b, b), (\mathsf{tid}_c, c)$ to $\mathsf{state}$.
   (e) Sample $\mathbf{ch} \xleftarrow{\$} \mathcal{T}$, append $(\mathsf{state} \| ccount, \mathbf{ch})$ to $\mathtt{mp}_{\mathrm{chal}}$ and send $(\mathbf{ch}, ccount)$ to $\mathcal{A}$.
   (f) If $\mathcal{A}$ sends $\mathsf{reject}$ then increase $ccount$ and go to step $a$. If $\mathcal{A}$ sends $(\mathsf{accept}, t)$ and $t \geq min$ find $(\mathsf{state}' \| t, \mathbf{ch})$ in $\mathtt{mp}_{\mathrm{chal}}$, add $(\mathsf{tid}_a, a), (\mathsf{tid}_b, b), (\mathsf{tid}_c, c)$ from $\mathsf{state}'$ to $\mathtt{mp}_{\mathrm{val}}$ and set $\mathtt{fl}_{\mathrm{ch}} = \mathbf{ch}$.
4. Send $(\mathsf{ShareTriple}, \mathsf{sid}, (\mathsf{tid}, a^i, b^i, c^i), \mathtt{fl}_{\mathrm{ch}})$ to each party $\mathcal{P}_i \in \mathcal{P} \setminus \tilde{\mathcal{P}}$.

Fig. 14: Ideal functionality $\mathcal{F}_{\mathsf{Offline}}$

## 6 Online Protocol

We present the protocol $\Pi_{\mathsf{Online}}$ in Figure 27 and Figure 28 for the online phase. The protocol uses $\mathcal{F}_{\mathsf{Offline}}$ to share inputs, create multiplication triples, perform linear operations and reconstruct shared values. Public messages are sent through

$\mathcal{F}_{\mathsf{Offline}}$ **(Part 2)**

**Linear** On input $(\mathsf{Linear}, \mathsf{sid}, \mathsf{id}_1, \mathsf{id}_2, \alpha, \delta)$ from all parties, where $(\mathsf{id}_1, v_1) \in \mathtt{mp}_{\mathrm{val}}$, $(\mathsf{id}_2, *) \notin \mathtt{mp}_{\mathrm{val}}$ and $\mathsf{id}_2 \neq \mathsf{rid}_*, \mathsf{tid}_*$, $\mathtt{fl}_{\mathrm{ver}} = \perp$ and $\alpha, \delta \in \mathbb{F}$: Set $v_2 = \alpha \cdot v_1 + \delta$ and store $(\mathsf{id}_2, v_2)$ in $\mathtt{mp}_{\mathrm{val}}$.
- On input $(\mathsf{Linear}, \mathsf{sid}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ from all parties, where $\{(\mathsf{id}_1, v_1)(\mathsf{id}_2, v_2)\} \subseteq \mathtt{mp}_{\mathrm{val}}$, $(\mathsf{id}_3, *) \notin \mathtt{mp}_{\mathrm{val}}$ and $\mathsf{id}_3 \neq \mathsf{rid}_*, \mathsf{tid}_*$ and $\mathtt{fl}_{\mathrm{ver}} = \perp$: Set $v_3 = v_1 + v_2$ and add $\mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3$

**Open** On input $(\mathsf{Open}, \mathsf{sid}, \mathsf{id}, \mathtt{type}, i)$ from all parties, if $\mathtt{fl}_{\mathrm{ver}} = \perp$, $(\mathsf{id}, v) \in \mathtt{mp}_{\mathrm{val}}$, $\mathtt{type} \in [0..2]$ and $i \in [0..n]$:
1. If $\mathtt{type} = 1$ or $\mathtt{type} = 0$ and $\mathcal{P}_i \in \tilde{\mathcal{P}}$ send $(\mathsf{Open}, \mathsf{sid}, \mathsf{id}, i, v)$ to $\mathcal{A}$.
2. Wait for $\mathcal{A}$ to send $\Delta \in \mathbb{F}, \delta \in \{0, 1\}$. If $\Delta \neq 0$ or $\delta = 0$ set $c = 0$, otherwise set $c = 1$.
3. Append $(\mathsf{id}, \mathtt{type}, v + \Delta, i, c)$ to $\mathtt{li}_{\mathrm{open}}$.
4. If $\mathtt{type} = 0$ and $\mathcal{P}_i \notin \tilde{\mathcal{P}}$ send $(\mathsf{Open}, \mathsf{sid}, i, v + \Delta)$ to $\mathcal{P}_i$.
5. If $\mathtt{type} = 1$ send $(\mathsf{Open}, \mathsf{sid}, i, v + \Delta)$ to all parties.
6. If $\mathtt{type} = 2$ send $(\mathsf{DOpen}, \mathsf{sid})$ to all parties.

**Release** On input $(\mathsf{Release}, \mathsf{sid})$ from all parties and if $\mathtt{fl}_{\mathrm{ver}} = \perp$: Set $\mathtt{fl}_{\mathrm{ver}} = 1$ and send $(\mathsf{Release}, \mathsf{sid})$ to each party.

**Full Release** On input $(\mathsf{Release}, \mathsf{sid})$ from all parties and if $\mathtt{fl}_{\mathrm{ver}} = 1$:
1. For each $(\mathsf{id}, 2, v, i) \in \mathtt{li}_{\mathrm{open}}$ send $(\mathsf{Release}, \mathsf{sid}, \mathsf{id}, v)$ to the adversary.
2. Upon input $(\mathsf{Ok}, \mathsf{sid})$ from the adversary, replace each $(\mathsf{id}, 2, v, i)$ on $\mathtt{li}_{\mathrm{open}}$ with $(\mathsf{id}, 1, v, i)$ and let $L$ be the list of all $(\mathsf{id}, v)$.
3. Set $\mathtt{fl}_{\mathrm{ver}} = 2$ and send $(\mathsf{Release}, \mathsf{sid}, L)$ to each party.

**Verify** On input $(\mathsf{Verify}, \mathsf{sid}, L)$ from $\mathcal{P}_i$ where for each $\mathsf{id} \in L$ there is a value $(\mathsf{id}, *, v, *, c) \in \mathtt{li}_{\mathrm{open}}$ and if $\mathtt{fl}_{\mathrm{ver}} \neq \perp$:
1. If $(\mathsf{id}, 1, *, *) \in \mathtt{li}_{\mathrm{open}}$ or $(\mathsf{id}, 0, *, i) \in \mathtt{li}_{\mathrm{open}}$ for each $\mathsf{id} \in L$ output $\mathsf{accept}$, otherwise $\mathsf{reject}$.

**Challenge** On input $(\mathsf{Challenge}, \mathsf{sid})$ by $\mathcal{P}_{\mathrm{A}}$ return $\mathtt{fl}_{\mathrm{ch}}$.

**Audit** On input $(\mathsf{Audit}, \mathsf{sid}, (\mathsf{id}_l, m_l, f_l, b_l)_{l \in [k]})$ from $\mathcal{P}_{\mathrm{A}}$ where $\mathtt{fl}_{\mathrm{ver}} = 2$ and each $f_l$ is a linear function in $\mathsf{tid}_j, \mathsf{rid}_j$:
1. If $(\mathsf{id}_l, 1, m_l, *) \in \mathtt{li}_{\mathrm{open}}$ for all $l \in [k]$ then return $\mathsf{accept}$, else $\mathsf{reject}$.

Fig. 15: Ideal functionality $\mathcal{F}_{\mathsf{Offline}}$

$\mathcal{F}_{\mathsf{GBB}}$ and randomness is created using the challenge mechanism of $\mathcal{F}_{\mathsf{Offline}}$. During the initialization triples and random values are created by $\mathcal{F}_{\mathsf{Offline}}$, and these are additively shared among the parties. For each input gate of the circuit one random sharing is needed and for each multiplication gate two triples will be created. Since triples may not be correct when returned by $\mathcal{F}_{\mathsf{Offline}}$, they are then verified using the **Triple Check** procedure. For each triple that is verified, one additional triple has to be consumed in the process.

To check correctness of a triple, we use the method due to [28]. During that procedure a random value $\tau$ is generated based on previous messages send to $\mathcal{F}_{\mathsf{GBB}}$. To verify one triple $x, y, z$ using another $a, b, c$, the parties compute and open $\rho = \tau \cdot x + a$ and $\sigma = y + b$. Using these, they compute and open

$$v = \tau \cdot z - c + \sigma \cdot a + \rho \cdot b - \sigma \cdot \rho$$

**$\Pi_{\mathsf{Offline}}$ (Part 1)**

The protocol runs among $n$ parties $\mathcal{P}$. All parties have access to the global subroutines $\mathcal{F}_{\mathsf{GpRO}}$ and $\mathcal{F}_{\mathsf{GBB}}$. Each $\mathcal{P}_i$ initializes an instance of $\mathcal{F}_{\mathsf{HOTCom}}$, such that it is the sender while every other party is in the set of receivers. We denote this instance as $\mathcal{F}^i_{\mathsf{HOTCom}}$, and each has challenge space $\{0,1\}^{2\lambda}$. Additionally each ordered pair of parties has one instance of $\mathcal{F}_{\mathsf{OLE}}$ running between them. When a party does not send an expected message then broadcast $(\mathsf{Broadcast}, \mathit{Abort})$ and stop the protocol. Each $\mathcal{P}_i$ sets $\mathtt{fl}_{\mathrm{coms}} = 0$ and $c = \bot$.

**Initialize** On input $(\mathsf{Init}, \mathsf{sid}, \mathsf{in}, \mathsf{m})$ every party $\mathcal{P}_i$ sends $(\mathsf{Init}, \mathsf{sid}, \mathsf{in} + 3\mathsf{m})$ to all instances of $\mathcal{F}_{\mathsf{HOTCom}}$.

**Input** On input $(\mathsf{Input}, \mathsf{sid})$ and if **Initialize** has been executed and $\mathtt{fl}_{\mathrm{coms}} = 0$, $\mathcal{P}_i$ does the following $\mathsf{in}$ times in parallel:

1. Sample $r^i \xleftarrow{\$} \mathbb{F}$ and send $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{rid}, r^i))$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$.
2. If all commits are finished, increase $\mathtt{fl}_{\mathrm{coms}}$ and output $(\mathsf{rid}_l, r^i_l)_{l\in[\mathsf{in}]}$.

**Triples** On input $(\mathsf{Triples}, \mathsf{sid})$, if **Initialize** has been executed and $\mathtt{fl}_{\mathrm{coms}} = 1$, $\mathcal{P}_i$ does the following $\mathsf{m}$ times in parallel:

1. For all $l \in [\mathsf{m}]$ sample $a^i, b^i$ uniformly at random and send $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{tid}_j, j^i)_{j\in\{a,b\}})$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$.
2. For each ordered set of parties $(\mathcal{P}_i, \mathcal{P}_j)$, $\mathcal{P}_i$ sends $(\mathsf{Extend}, \mathsf{sid}, a^i)$ and $\mathcal{P}_j$ sends $(\mathsf{Extend}, \mathsf{sid}, b^j)$ to $\mathcal{F}_{\mathsf{OLE}}$. $\mathcal{P}_i$ receives $w^{i,j} = a^i \cdot b^j - v^{i,j}$ while $\mathcal{P}_j$ receives $v^{i,j}$.
3. Repeat the aforementioned step with inputs $(\mathsf{Extend}, \mathsf{sid}, a^j)$ from $\mathcal{P}_j$ and $(\mathsf{Extend}, \mathsf{sid}, b^i)$ from $\mathcal{P}_i$. $\mathcal{P}_j$ obtains $w^{i,j}$ while $\mathcal{P}_i$ gets $v^{i,j}$.
4. For each $l \in [\mathsf{m}]$ party $\mathcal{P}_i$ computes $c^i = a^i \cdot b^i + \sum_{j\neq i}^{n} w^{i,j} + v^{i,j}$
5. Each party $\mathcal{P}_i$ sends $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{tid}_c, c^i))$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$.
6. From the last commit response, each party $\mathcal{P}_i$ obtains the challenge value $\overline{\mathbf{ch}}^j$ from each $\mathcal{F}^j_{\mathsf{HOTCom}}$.
7. Send $(\mathsf{Query}, \mathsf{sid}, (\overline{\mathbf{ch}}^1 || \ldots || \overline{\mathbf{ch}}^{|P|}), \mathbb{F}_p^*)$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain $\mathbf{ch}$.
8. Output $((\mathsf{tid}_l, a^i_l, b^i_l, c^i_l, l)_{l\in[\mathsf{m}]}, \mathbf{ch})$. Increase $\mathtt{fl}_{\mathrm{coms}}$.

**Linear** On input $(\mathsf{Linear}, \mathsf{sid}, \mathsf{id}_1, \mathsf{id}_2, \alpha, \beta)$ where $\alpha, \beta \in \mathbb{F}$ $\mathcal{P}_i$ sends $(\mathsf{Linear}, \mathsf{sid}, \mathsf{id}_1, \mathsf{id}_2, \alpha, \beta)$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$. On input $(\mathsf{Linear}, \mathsf{sid}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$, $\mathcal{P}_i$ sends $(\mathsf{Add}, \mathsf{sid}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$.

Fig. 16: Protocol $\Pi_{\mathsf{Offline}}$ to implement the preprocessing phase.

which is expected to be 0. If the triples are not correct, then $v$ can falsely open to 0 with probability $\frac{1}{|\mathbb{F}|-1}$ as explained in [28]. Once the triple check was successful, parties can share their inputs using **Input**. For any $\mathcal{P}_i$ to share a value $x$, they will require a random value $r$, which was created during **Init** and is privately opened to $\mathcal{P}_i$. $\mathcal{P}_i$ will then send $\delta = x + r$ to $\mathcal{F}_{\mathsf{GBB}}$, allowing parties to publicly correct the sharing of $r$ to $x$.

To evaluate the circuit $\mathcal{C}$ the parties use the **Linear** and **Multiply** commands. As the shares have linear properties, computing linear gates of $\mathcal{C}$ is straightforward. For multiplication we use multiplication triples as proposed by Beaver [11]: When $x, y$ and $a, b, c$ are additively shared and $c = a \cdot b$ then $z = \alpha \cdot b + \beta \cdot a - \alpha \cdot \beta + c$

$\Pi_{\mathsf{Offline}}$ **(Part 2)**

**Open** On input $(\mathsf{Open}, \mathsf{sid}, \mathsf{id}, \mathtt{type}, j)$ where $\mathsf{id}$ was committed to using $\mathcal{F}_{\mathsf{HOTCom}}$, $\mathtt{type} \in [0..2]$ and $j \in [0..n]$, $\mathcal{P}_i$ does the following:
- If $\mathtt{type} \in \{0, 1\}$ send $(\mathsf{Hint}, \mathsf{sid}, \mathsf{id}, j)$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$.
- If $\mathtt{type} = 1$ or $\mathtt{type} = 0$ and $i = j$ then obtain $(\mathsf{Hint}, \mathsf{sid}, \mathsf{id}, v^j)$ from all $\mathcal{F}^j_{\mathsf{HOTCom}}$, compute $v = \sum_{j=1}^{n} v^j$ where $v^i$ is the value $\mathcal{P}_i$ committed to for $\mathsf{id}$ in $\mathcal{F}^i_{\mathsf{HOTCom}}$ and output $v$.
- If $\mathtt{type} = 2$ send $(\mathsf{DelHint}, \mathsf{sid}, \mathsf{id})$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$ and wait for $(\mathsf{DelHint}, \mathsf{sid}, \mathsf{id})$ from every $\mathcal{F}^j_{\mathsf{HOTCom}}$.

**Release** On input $(\mathsf{Release}, \mathsf{sid})$ each $\mathcal{P}_i$ sends $(\mathsf{Open}, \mathsf{sid})$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$. If any $\mathcal{F}^i_{\mathsf{HOTCom}}$ outputs $(\mathsf{Abort}, \mathsf{sid})$ then abort.

**Full Release** On input $(\mathsf{Release}, \mathsf{sid})$ each party $\mathcal{P}_i$ does the following:
1. Send $(\mathsf{Open}, \mathsf{sid})$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$. If any $\mathcal{F}^j_{\mathsf{HOTCom}}$ aborts then abort.
2. Upon obtaining $(\mathsf{Open}, \mathsf{sid}, L^j)$ from each $\mathcal{F}^j_{\mathsf{HOTCom}}$ where $L^j = \{(\mathsf{id}_l, v^j_l)\}_{l \in [k]}$, set $v_l = \sum_{j=1}^{n} \mathsf{id}^j_l$
3. Output $\{(\mathsf{id}_l, v_l)\}_{l \in [k]}$.

**Verify** On input $(\mathsf{Verify}, \mathsf{sid}, (\mathsf{cid}_l)_{l \in [k]}, h)$, $\mathcal{P}_i$ does the following:
1. Let $v^j_l$ be the shares of $\mathsf{cid}_l$ that $\mathcal{P}_i$ obtained previously. If any $v^j_l$ is undefined then output $\mathsf{reject}$.
2. Send $(\mathsf{Verify}, \mathsf{sid}, (\mathsf{cid}_l, v^j_l)_{l \in [k]})$ to each $\mathcal{F}^j_{\mathsf{HOTCom}}$.
3. If any $\mathcal{F}^j_{\mathsf{HOTCom}}$ returns $(\mathsf{VerifyFailed}, \mathsf{sid})$ then output $\mathsf{reject}$.

**Audit** On input $(\mathsf{Audit}, \mathsf{sid}, (\mathsf{cid}_l, m_l, f_l)_{l \in [k]}, h)$, $\mathcal{P}_{\mathrm{A}}$ does the following:
1. Send $(\mathsf{Audit}, \mathsf{sid}, (\mathsf{cid}_l, f_l)_{l \in [k]})$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$. If any $\mathcal{F}^i_{\mathsf{HOTCom}}$ returns $(\mathsf{AuditFailed}, \mathsf{sid})$ output $\mathsf{reject}$.
2. If all return $(\mathsf{Audit}, \mathsf{sid}, L^j)$ with $L^j = (\mathsf{id}_l, v^j_l)$ then set $m'_l = \sum_{j=1}^{n} v^j_l$. If any $m_l \neq m'_l$ output $\mathsf{reject}$, otherwise output $\mathsf{accept}$.

**Challenge** On input $(\mathsf{Challenge}, \mathsf{sid})$ to $\mathcal{P}_{\mathrm{A}}$ send $(\mathsf{Challenges}, \mathsf{sid})$ to every $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance. In the returned list take the most recent challenge that is associated with **Triples** for each $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance $\overline{\mathbf{ch}}^i$. Concatenate all $\overline{\mathbf{ch}}^i$ values and query $\mathcal{F}_{\mathsf{GpRO}}$, output what it outputs.

Fig. 17: Protocol $\Pi_{\mathsf{Offline}}$ to implement the preprocessing phase.

is equivalent to $z = x \cdot y$ when computing and opening $\alpha = x - a$ and $\beta = y - b$. For all linear computations, the correctness of openings can be verified using $\mathcal{F}_{\mathsf{Offline}}$, hence our computation yields a correct $y$ if no party cheated. Before outputting the result of the circuit, **Verify** is called to check if the adversary cheated in the computation or during openings (thus implying correctness of triples). After **Output** was successful and $y$ is public, any auditor $\mathcal{P}_{\mathrm{A}}$ can use **Audit** of $\mathcal{F}_{\mathsf{Offline}}$ to verify the the computation. The full protocol can be found in 27 and 28 in Appendix H, as well as the security proof.

**Theorem 4.** *In the $(\mathcal{F}_{\mathsf{GBB}}, \mathcal{F}_{\mathsf{Offline}})$-hybrid model, the protocol $\Pi_{\mathsf{Online}}$ implements $\mathcal{F}_{\mathsf{Online}}$ with statistical security against any static active adversary. Assuming the adversary sends* reject *or* reset *at most $Q$ times to $\mathcal{F}_{\mathsf{Offline}}$, the distinguishing advantage is at most $\frac{Q}{|\mathbb{F}|-1}$.*

## References


1. Adida, B.: Helios: Web-based open-audit voting. In: van Oorschot, P.C. (ed.) USENIX Security 2008. pp. 335–348. USENIX Association (Jul / Aug 2008), http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf
2. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2087–2104. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134104
3. Badertscher, C., Canetti, R., Hesse, J., Tackmann, B., Zikas, V.: Universal composition with global subroutines: Capturing global setup within plain UC. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part III. LNCS, vol. 12552, pp. 1–30. Springer, Cham (Nov 2020). https://doi.org/10.1007/978-3-030-64381-2_1
4. Baldimtsi, F., Kiayias, A., Zacharias, T., Zhang, B.: Crowd verifiable zero-knowledge and end-to-end verifiable multiparty computation. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 717–748. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64840-4_24
5. Baum, C., Beullens, W., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: Shorter, tighter, FAESTer: Optimizations and improved (QROM) analysis for VOLE-in-the-head signatures. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part VI. LNCS, vol. 16005, pp. 124–156. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01887-8_5
6. Baum, C., Beullens, W., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 463–493. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_15
7. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 581–615. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_19
8. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. In: Abdalla, M., De Prisco, R. (eds.) SCN 14. LNCS, vol. 8642, pp. 175–196. Springer, Cham (Sep 2014). https://doi.org/10.1007/978-3-319-10879-7_11
9. Baum, C., Dittmer, S., Scholl, P., Wang, X.: Sok: vector OLE-based zero-knowledge protocols. DCC **91**(11), 3527–3561 (2023). https://doi.org/10.1007/s10623-023-01292-8
10. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 92–122. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_4
11. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 420–432. Springer, Berlin, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_34
12. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 701–732. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_23

13. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Berlin, Heidelberg (May 2011). https://doi.org/10.1007/978-3-642-20465-4_11
14. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Berlin, Heidelberg (Feb 2009). https://doi.org/10.1007/978-3-642-03549-4_20
15. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-LPN. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 387–416. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_14
16. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
17. Canetti, R., Chen, M.: Universally composable succinct vector commitments and applications. In: Theory of Cryptography Conference. pp. 642–674. Springer (2025)
18. Cascudo, I., Damgård, I., David, B., Döttling, N., Dowsley, R., Giacomelli, I.: Efficient UC commitment extension with homomorphism for free (and applications). In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 606–635. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-34621-8_22
19. Cunningham, R.K., Fuller, B., Yakoubov, S.: Catching MPC cheaters: Identification and openability. In: Shikata, J. (ed.) ICITS 17. LNCS, vol. 10681, pp. 110–134. Springer, Cham (Nov / Dec 2017). https://doi.org/10.1007/978-3-319-72089-0_7
20. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Berlin, Heidelberg (Sep 2013). https://doi.org/10.1007/978-3-642-40203-6_1
21. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_38
22. Frederiksen, T.K., Pinkas, B., Yanai, A.: Committed MPC - maliciously secure multiparty computation from homomorphic commitments. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 587–619. Springer, Cham (Mar 2018). https://doi.org/10.1007/978-3-319-76578-5_20
23. Gilboa, N.: Two party RSA key generation. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 116–129. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_8
24. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 830–842. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978357
25. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 158–189. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_6

26. Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., van der Maaten, L.: Crypten: Secure multi-party computation meets machine learning. Advances in Neural Information Processing Systems **34**, 4961–4973 (2021)
27. Küsters, R., Liedtke, J., Müller, J., Rausch, D., Vogt, A.: Ordinos: A verifiable tally-hiding E-voting system. In: 2020 IEEE European Symposium on Security and Privacy. pp. 216–235. IEEE Computer Society Press (Sep 2020). https://doi.org/10.1109/EuroSP48549.2020.00022
28. Lindell, Y., Nof, A.: A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 259–276. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133999
29. Nair, D.G., Binu, V., Kumar, G.S.: An improved e-voting scheme using secret sharing based secure multi-party computation. arXiv preprint arXiv:1502.07469 (2015)
30. Rachuri, R., Scholl, P.: Le mans: Dynamic and fluid MPC for dishonest majority. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 719–749. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15802-5_25
31. Rivinius, M., Reisert, P., Rausch, D., Küsters, R.: Publicly accountable robust multi-party computation. In: 2022 IEEE Symposium on Security and Privacy. pp. 2430–2449. IEEE Computer Society Press (May 2022). https://doi.org/10.1109/SP46214.2022.9833608
32. Schoenmakers, B., Veeningen, M.: Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) ACNS 2015. LNCS, vol. 9092, pp. 3–22. Springer, Cham (Jun 2015). https://doi.org/10.1007/978-3-319-28166-7_1
33. Shoup, V.: A computational introduction to number theory and algebra. Cambridge university press (2009)
34. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: 2021 IEEE Symposium on Security and Privacy. pp. 1074–1091. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00056
35. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2986–3001. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484556

## A Open problems

While we give the first construction of plausibly post-quantum verifiable MPC from generic assumptions (the existence of OLE and a random oracle) and in fact the first MPC protocol that uses VOLEitHfor MACs, there are multiple shortcomings which may lead to interesting future work.

*No support for binary circuits.* Our construction relies on an MPC scheme which is only secure for fields $\mathbb{F}$ of size exponential in the security parameter. While this is an improvement over [8], it would be interesting to extend our techniques to polynomial-size fields or even $\mathbb{F}_2$. Note that our one-time linear commitment

scheme can be adapted to this setting by using techniques from [7] and the dependence on exponential field size mainly stems from the [30] protocol.

*Client-server setting with one message.* One interesting property of [8] is that it allows a client-server MPC use, where clients can give input to the MPC protocol and must only send one message to the servers. This crucially relies on the clients being able to check commitment openings for correctness, which is not possible with our *hint* mechanism in just one round. It would therefore be interesting if our protocol can be adapted to the client-server setting with one-round client communication.

*Identifiable abort.* An interesting, related property to public verifiability is to let MPC parties deduce who cheated in a MPC protocol in case of abort. This is also called *identifiable abort*. In this work, we focus on public verifiability as the main security goal. We believe that our construction could be modified to also permit identifiable abort, e.g. by including an identifiable triple generation protocol. However, this is beyond the scope of this work.

# B Extended Preliminaries

## B.1 Universal Hash functions

A universal hash function family $\mathcal{H}$ is an efficiently samplable set of functions such that, upon choosing two different inputs $x, y$ it holds that $h(x) = h(y)$ with bounded probability for a randomly chosen $h \in \mathcal{H}$.

**Definition 1 (Universal Hash function).** *Let $\mathcal{H} = \{h : S \to T\}$, where $S, T$ are finite sets, then $\mathcal{H}$ is a family of $\epsilon$-universal hash functions if*

$$\forall x, y \in S, x \neq y : \Pr_{h \in_R \mathcal{H}}[h(x) = h(y)] \leq \epsilon$$

The family of $1/p$-universal hash functions used in the work is defined as:

**Proposition 2.** *Let $p$ be a prime and $l$ be a positive integer. Then for every input $r = (r_1, \ldots, r_{n+1}) \in \mathbb{F}_p^{l+1}$, the function*

$$h_{\mathbf{s}} : \mathbb{F}_p^l \to \mathbb{F}_p\,, \qquad (r_1, \ldots, r_{l+1}) \mapsto r_{n+1} + \sum\nolimits_{i=1}^{l} r_i \dot{s}_i$$

*is a $1/p$-universal hash function according to Definition 1.*

*Proof.* See e.g. [33, Example 8.37].

In this work, we use universal hash functions that are hiding.

**Definition 2 (Hiding).** *A matrix* $\mathbf{H} \in \mathbb{F}_p^{r \times (n+h)}$ *is* $n$*-hiding if the distribution of* $\mathbf{Hv}$ *is independent from* $\mathbf{v}_{[1..n]}$ *when* $\mathbf{v}_{[n+1..n+h]}$ *is chosen uniformly at random from* $\mathbb{F}_p^h$*. A hash family* $\mathcal{H}$ *is* $n$*-hiding if every* $\mathbf{H} \in \mathcal{H}$ *is* $n$*-hiding.*

The construction in Proposition 2 fulfills this property for $r = h = 1$ if the value $\mathbf{v}_{n+1}$ is chosen uniformly at random.

### B.2 The UC Framework

We use a version of the universal composability (UC) framework [16] that respects global subroutines [3] for analyzing security of our protocols.

In UC, protocols are run by interactive Turing Machines (iTMs) called *parties* which we denote with a $\mathcal{P}$. The *adversary* $\mathcal{A}$, which is also an iTM, can actively corrupt any of these parties and then gains full control over it, seeing messages that it obtains and choosing any messages that the corrupted party or parties send. In this work we only consider static corruptions, meaning that the choice of corruption is fixed from the beginning of the protocol.

The parties can communicate with each other directly and with so-called *ideal functionalities* (which themselves are iTMs). These ideal functionalities are denoted by $\mathcal{F}$ and abstractly describe cryptographic protocols with certain guarantees. As usual, we define security with respect to an iTM $\mathcal{Z}$ called the *environment*. The environment provides inputs to the parties, receives outputs from them and controls $\mathcal{A}$.

To define security, let $\pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ be the distribution of the output of an arbitrary $\mathcal{Z}$ when interacting with $\mathcal{A}$ in a real protocol instance $\pi$ using ideal functionalities $\mathcal{F}_1, \ldots$. Furthermore, let $\mathsf{Sim}$ denote an *ideal world adversary* and $\mathcal{F} \circ \mathsf{Sim}$ be the distribution of the output of $\mathcal{Z}$ when interacting with parties which run with $\mathcal{F}$ instead of $\pi$ and where $\mathsf{Sim}$ takes care of adversarial behavior.

**Definition 3.** *We say that* $\pi$ *UC-securely implements* $\mathcal{F}$ *if for every iTM* $\mathcal{A}$ *there exists an iTM* $\mathsf{Sim}$ *(with black-box access to* $\mathcal{A}$*) such that for every environment* $\mathcal{Z}$*, the outputs of* $\mathcal{Z} \circ \pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ *and* $\mathcal{Z} \circ \mathcal{F} \circ \mathsf{Sim}$ *are identical except with negligible probability.*

**UC with global subroutines.** While UC allows to analyze security of protocols under arbitrary composition, it falls short of modeling global routines that maintain a shared state for multiple sessions - such as random oracles or bulletin boards. Therefore, we analyze our work in the *Universal Composition with Global Subroutines* (UCGS) model, proposed in [3].

Let $\pi$ be a protocol that makes use of a global subroutine $\mathcal{G}$ (or more such subroutines). UCGS describes how to transform $\pi, \mathcal{G}$ into a standard UC-protocol $M[\pi, \mathcal{G}]$. This gives $\pi, \mathcal{F}$ and $\mathcal{Z}$ access to $\mathcal{G}$ such that it UC-emulates $M[\mathcal{F}, \mathcal{G}]$. The global subroutine $\mathcal{G}$ has to fulfill the *regularity* condition: a subroutine is

*regular* if only previously registered machines can speak with it and if it does not activate other machines outside of $\pi$. Moreover, the subroutine is not allowed to spawn any new instances of the protocol calling it.

**Definition 4.** *We say that $\pi$ implements $\mathcal{F}$ in the presence of a global subroutine $\mathcal{G}$ if for any $\mathcal{A}$ there exists an iTM $\mathcal{S}$ such that for every environment $\mathcal{Z}$ the outputs of an execution $EXEC^{G}_{\pi,\mathcal{A},\mathcal{Z}}$ are identical to an execution $EXEC^{G}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ except with negligible probability.*

More details, as well as a composition theorem for UCGS and further examples, can be found in [3].

## C Communication Complexity

We estimate the communication complexity and divide the communication cost in offline and online phase. Communication is measured on a per-sender basis. As sending messages to $\mathcal{F}_{\mathsf{GBB}}$ is different from sending peer-to-peer messages we classify communication cost according to the used channel. We further assume, that all $n$ parties send their messages simultaneously and (if possible) each party will bundle their messages to one message instead of sending separate messages in consecutive rounds. We leave out *compressible protocol management information* that we currently send along through $\mathcal{F}_{\mathsf{GBB}}$ - this includes terms like $\mathsf{chall}$, commitment IDs and indices of functions on commitments such as $f_{id}$. As before, we assume that the circuit has $\mathsf{in}$ input gates and $\mathsf{m}$ multiplication gates.

*Broadcast Offline Phase.* Each pair of parties has to use **Init** in $\mathcal{F}_{\mathsf{WVOLE}}$ both as sender and receiver where a total of $\mathrm{mc} = \mathsf{in} + 6\mathsf{m} + 1$ commitments must be supported. Towards instantiating $\mathcal{F}_{\mathsf{HOTCom}}$ we assume that $k_{\mathcal{C}} = n_{\mathcal{C}} = \mathcal{O}(\lambda)$ and we set $N = \mathcal{O}(\lambda)$ asymptotically as well. During initialization, $\Pi_{\mathsf{IndWVole}}$ broadcasts $(n-1) \cdot 2\lambda$ bits (for $H$ for each receiver) while $\Pi_{\mathsf{CodeWVole}}$ sends

$$(n-1)\log(p) \cdot ((2\ell+1)(n_{\mathcal{C}} - k_{\mathcal{C}}) + n_{\mathcal{C}} + k_{\mathcal{C}})$$

to communicate $\mathbf{C}, \widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}$ where $\ell = \lceil \mathrm{mc}/k_{\mathcal{C}} \rceil$. This is achieved in 3 broadcast rounds. During the offline protocol all input values as well as the $a, b$ values of the Beaver triples can be generated and committed to simultaneously. Together with the initial broadcast of the circuit $\mathcal{C}$ and the finalization of the triple generation our offline phase can be instantiated with 3 broadcast rounds. In them, each party broadcasts

$$(n-1) \cdot \mathrm{mc} \cdot \log(p)$$

bits to commit to the triples as well as input blindings and the blinding used in the consistency check of $\Pi^{1-n}_{\mathsf{HOTCommit}}$. Table 2 gives an overview of the amount of bits that has to be send over broadcast as well as the number of rounds for each of these operations.

*Broadcast Online Phase.* The online phase includes (triple) verification, the input of secret values for all parties, the circuit evaluation and the output, which includes verification and opening of $\mathcal{F}_{\mathsf{WVOLE}}$. We assume that in the circuit $\mathcal{C}$ a maximum of $\mathcal{W}$ multiplications can be performed in parallel and the depth $\mathcal{D}$ is the maximum number of multiplications that have to be performed consecutively. The online phase can be completed in $4 + \mathcal{D}$ rounds, the details for each phase and number of bits can be found in Table 3.

To verify the triples, each party has to communicate 4 openings which amounts to $4\mathsf{m}\log(p)$ bits of broadcast. During the input phase, each party broadcasts $(n-1) \cdot \mathsf{in} \cdot 2\lambda$ bits for commitments used in private openings. To then evaluate the circuit, each party must broadcast $2\mathsf{m}\log(p)$ bits to perform multiplications using Beaver's circuit randomization technique. During the output phase the final value $y$ is sent to the future using **Delayed Hint** for which a total of $(n-1)\cdot 2\lambda + 2\lambda$ have to be sent via $\mathcal{F}_{\mathsf{GBB}}$. The consistency check that is performed in $\Pi^{1-n}_{\mathsf{HOTCommit}}$ before the release of previously opened values amounts to only $\log(p)$ bits. For the final release of $y$ a total of $(n-1) \cdot 2\log(p) + \lambda$ bits are send to the broadcast channel.

More dominant is the communication part of **Open** in $\Pi^{1-1}_{\mathsf{HOTCommit}}$, where each instance must broadcast $A_{\mathtt{pub}}, A_{\mathtt{delay}}, \mathbf{S}, \mathbf{D}$ which communicates

$$(n-1) \cdot \log(p) \cdot (\ell \cdot k_{\mathcal{C}} + \ell \cdot n_{\mathcal{C}} + (\mathsf{in} + 5\mathsf{m})) + n \cdot 2\lambda$$

bits via the broadcast channel. In $\mathcal{F}_{\mathsf{WVOLE}}$ the main communication stems from **Deliver** and **InitDone** run by $\Pi_{\mathsf{IndWVole}}$ where we broadcast the $\mathbf{t}_{i,j}, \mathbf{r}_{i,j}, c_{i,j}$ values which amount to

$$(n-1) \cdot 2n_{\mathcal{C}} \cdot (\lambda \cdot N + (N-1) \cdot \ell \cdot \log(p))$$

bits of communication. The overall amount of bits during **Output** sum up to

$$\begin{aligned}&(2n-1) \cdot 2\lambda + 3\lambda + \log(p) + (n-1) \cdot 2\log(p)\\ &+(n-1) \cdot \log(p) \cdot (\ell \cdot k_{\mathcal{C}} + \ell \cdot n_{\mathcal{C}} + (\mathsf{in} + 5\mathsf{m}))\\ &+(n-1) \cdot 2n_{\mathcal{C}} \cdot (\lambda \cdot N + (N-1) \cdot \ell \cdot \log(p)).\end{aligned}$$

*Peer-to-peer communication.* All communication that is peer-to-peer comes from $\mathcal{F}_{\mathsf{OLE}}$ and private openings done in $\mathcal{F}_{\mathsf{HOTCom}}$. As our construction is flexible in which schemes to use for implementing $\mathcal{F}_{\mathsf{OLE}}$ an exact number of rounds as well as the length of messages can not be given. In the offline phase each pair of parties uses **Extend** in $\mathcal{F}_{\mathsf{OLE}}$ twice, where they extend $u, x \in \mathbb{F}^{\ell}$ to $v, w \in \mathbb{F}^{\ell}$. OLE can be implemented in communication sublinear in $\ell$ [15] in a constant number of rounds. The opening of the $\mathsf{in}$ input commitments via **Hint** sends a total of $\mathsf{in} \cdot \log(p)$ bits with another $\mathsf{in} \cdot (\log(p) + 2\lambda)$ bits during **Open**.

*Cost of auditability.* To examine the costs of adding public audibility to the underlying MPC protocol, we compare our work to [30]. Their construction is secure

in the dishonest majority setting and supports dynamic switching of committees. For the comparison we assume that no switching is performed. This paper is a prime candidate for comparison as the offline phase in our construction is quite similar to the one in [30]. The first source of additional cost is the need for a bulletin board rather than a standard broadcast channel. In addition, there are three further sources of overhead. First, the matrices for the linear codes used in $\mathcal{F}_{\mathsf{HOTCom}}$ must be communicated. Their dimensions are at most $\ell \cdot n_{\mathcal{C}} = O(|C|)$. Second, instead of using a global MAC key for all required values (as in [30]), our construction requires a separate commitment for each value. Third, compared to the [30] approach, where private openings are sent only to the receiving party, each party must additionally post a commitment to the bulletin board, which also applies to delayed openings in a similar way.

In terms of triple verification [30] and our construction use different approaches but both require 6 openings in total. In their approach the $c = a \cdot b$ value has to be authenticated using one opening and every gate in the circuit is evaluated once using the actual value and once using a randomized value, consuming two triples for each multiplication. In our approach we use the opening procedure from [28]. [30] can be realized in different ways, combining their preprocessing with the SPDZ implementation of ([25], [24]) where triples are verified during the offline phase yielding the most efficient online communication. When using this implementation the communication during the preprocessing phase are $O(|C|)+O(n\log(|C|))$ field elements, while the communication during the online phase can be as low as $2|C|$ field elements. This however ignores that the preprocessing phase uses programmable PCGs whose communication is dependent on the security parameter ($\lambda$). When the triple verification is not performed in the offline phase and their dynamic online phase is used (allowing one switch of committees between offline and online phase) the communication complexity in the preprocessing phase is lower with $O(n\log|C|)$ field elements, while the online phase communication is higher with $6|C|$ field elements. Also ignoring the programmable PCGs implementation.
Our approach requires $O(n\cdot\lambda\cdot|C|)$ bits to be communicated per party during the preprocessing phase while our triple verification requires $n(3\cdot\mathsf{m})$ field elements. The remainder of our online phase requires roughly $|C|\cdot n$ field elements and additional $O(n\cdot\lambda^2\cdot|C|)$ bits where most of the overhead is due to the commitment openings. This can trivially be lowered to $O(n\cdot\lambda\cdot\log(\lambda)\cdot|C|)$ by generating the $\mathbf{t}_{i,j}, r_{i,j}$ in $\Pi_{\mathsf{IndWVole}}$ as output of a GGM tree and only revealing seeds of the opened paths, as done in [7,6].

## D Constructing $\mathcal{F}_{\mathsf{WVOLE}}$ for Theorem 1

We will now outline how to construct $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ with parameters as required for $\mathcal{F}_{\mathsf{HOTCom}}$. As in [7] we will achieve this in two steps: first, we build an intermediate $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_{\mathcal{C}}}}$ functionality where $k_{\mathcal{C}} = n_{\mathcal{C}}$, i.e. all columns of $\mathbf{U}$ are random and independent. Then, using $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_{\mathcal{C}}}}$ we build the linear-code $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ by letting

| Offline | Total bits | Big-O (bits) | Rounds |
|---|---|---|---|
| $\mathcal{C}, H$ | $\lvert\mathcal{C}\rvert + (n-1)\cdot 2\lambda$ | $n \cdot \lambda$ | 1 |
| $\mathbf{C}, \widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}$, $r_{cc}$ | $(n-1)\log(p)\cdot((2\ell+1)(n_{\mathcal{C}}-k_{\mathcal{C}})$ $+n_{\mathcal{C}}+k_{\mathcal{C}}+1)$ | $n\cdot\log(p)\cdot\ell\cdot\lambda$ | 2 |
| $\{r_j\}_{j\in\mathsf{in}}$, $\{a_k, b_k\}_{k\in 2\mathsf{m}}$ | $(\mathsf{in}+4\mathsf{m})\cdot(n-1)\cdot\log(p)$ | $n(\mathsf{in}+\mathsf{m})\cdot\log(p)$ | 1 |
| $\{c_k\}_{k\in 2\mathsf{m}}$ | $(2\mathsf{m})\cdot(n-1)\cdot\log(p)$ | $n\cdot\mathsf{m}\cdot\log(p)$ | 1 |

Table 2: Overview of offline phase communication through $\mathcal{F}_{\mathsf{GBB}}$, where only $\mathcal{P}_1$ sends $\mathcal{C}$. Bits are counted for one party, the commitment of $r_{cc}$ is used for the consistency check in $\Pi^{1-n}_{\mathsf{HOTCommit}}$.

| Online | Total bits | Big-O (bits) | Rounds |
|---|---|---|---|
| **Triple Verify** | $4(n-1)\cdot\log(p)+2(n-1)\cdot 2\lambda$ | $n(\log(p)+\lambda)$ | 1 |
| **Input** | $\mathsf{in}\cdot\log(p)$ | $\mathsf{in}\cdot\log(p)$ | 1 |
| **Multiply** | $\mathcal{W}\cdot 2n(\log(p))$ | $(\mathcal{W}\cdot n)\cdot\log(p)$ | $\mathcal{D}$ |
| **Output** | $(2n-1)\cdot 2\lambda+3\lambda+\log(p)+(n-1)\cdot 2\log(p)$ $(n-1)\cdot\log(p)\cdot(\ell\cdot k_{\mathcal{C}}+\ell\cdot n_{\mathcal{C}}+(\mathsf{in}+5\mathsf{m}))$ $+(n-1)\cdot 2n_{\mathcal{C}}\cdot(\lambda\cdot N+(N-1)\cdot\ell\cdot\log(p))$ | $n\cdot\lambda(\lvert\mathcal{C}\rvert+\lambda^2$ $+\lambda\cdot\ell\cdot\log(p))$ | 2 |

Table 3: Overview of online phase communication through $\mathcal{F}_{\mathsf{GBB}}$. Bits are counted for one party.

$\mathcal{P}_{\mathrm{S}}$ correlate the columns of $\mathbf{U}$ according to $\mathcal{C}$ row-wise and using a consistency check to guarantee that the correlation is correct.

### D.1 Building $\mathcal{F}^{p,N,\mathbb{F}_p^{n_{\mathcal{C}}}}_{\mathsf{WVOLE}}$

We now describe the protocol $\Pi_{\mathsf{IndWVole}}$, which implements $\mathcal{F}^{p,N,\mathbb{F}_p^{n_{\mathcal{C}}}}_{\mathsf{WVOLE}}$. Our construction is very simple and uses the random oracle model to construct commitments, where we build subspace VOLE from a batch vector commitment. It also is highly unoptimized with respect to communication, as we mainly focus on making the protocol from [7] auditable. Any optimizations for smaller vector commitments and openings from [7] and other works also apply to our protocol.

$\Pi_{\mathsf{IndWVole}}$

We let $N = \mathsf{poly}(\lambda), N < p$. Let $n_{\mathcal{C}}$ be given. The protocol runs between $\mathcal{P}_{\mathrm{S}}, \mathcal{P}_{\mathrm{R}}, \mathcal{P}_{\mathrm{A}}$ and has access to global functionalities $\mathcal{F}_{\mathsf{GpRO}}, \mathcal{F}_{\mathsf{GBB}}$. Let $\mathcal{T} = \{0,1\}^{2\lambda}$ be the challenge space.

**Init** On $(\mathsf{Init}, \mathsf{sid}, \ell)$, $\mathcal{P}_{\mathrm{S}}$ does the following:

1. For $i \in [N], j \in [n_{\mathcal{C}}]$, sample $\mathbf{t}_{i,j} \xleftarrow{\$} \mathbb{F}_p^{\ell}$ and $r_{i,j} \xleftarrow{\$} \{0,1\}^{\lambda}$.
2. For each $j \in [n_{\mathcal{C}}]$ compute $\mathbf{u}_j := \sum_{i\in[N]} \mathbf{t}_{i,j}$ and $\mathbf{v}_j := -\sum_{i\in[N]} i \cdot \mathbf{t}_{i,j}$.
3. For each $i \in [N], j \in [n_{\mathcal{C}}]$ send $(\mathsf{Query}, \mathsf{sid}, \mathbf{t}_{i,j} \| r_{i,j}, \{0,1\}^{2\lambda})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain $c_{i,j}$. Then send $(\mathsf{Query}, \mathsf{sid}, \ell \| (c_{i,j})_{i\in[N]}^{j\in[n_{\mathcal{C}}]}, \{0,1\}^{2\lambda})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain $H$ and send $(\mathsf{Query}, \mathsf{sid}, \ell \| H, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain $h \in \mathcal{T}$.
4. Set $\mathbf{U} = [\mathbf{u}_1 \| \cdots \| \mathbf{u}_{n_{\mathcal{C}}}]$ and $\mathbf{V} = [\mathbf{v}_1 \| \cdots \| \mathbf{v}_{n_{\mathcal{C}}}]$. Then broadcast $(\ell, H)$ and output $(\mathbf{U}, \mathbf{V}, h)$.

Upon seeing $(\ell, H)$ on $\mathcal{F}_{\mathsf{GBB}}$, $\mathcal{P}_{\mathrm{R}}$ sends $(\mathsf{Query}, \mathsf{sid}, \ell \| H, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and denotes the response as $h \in \mathcal{T}$. Then it outputs $(\mathsf{Init}, \mathsf{sid}, \ell, h)$.

**Init Done** Upon input $(\mathsf{InitDone}, \mathsf{sid}, m)$, $\mathcal{P}_{\mathrm{S}}$ does the following:

1. $\mathcal{P}_{\mathrm{S}}$ broadcasts $(m, (c_{i,j})_{i\in[N]}^{j\in[n_{\mathcal{C}}]})$.
2. $\mathcal{P}_{\mathrm{R}}$ sends $(\mathsf{Query}, \mathsf{sid}, \ell \| (c_{i,j})_{i\in[N]}^{j\in[n_{\mathcal{C}}]}, \{0,1\}^{2\lambda})$ to $\mathcal{F}_{\mathsf{GpRO}}$. If the result differs from $H$ then it aborts.
3. Both $\mathcal{P}_{\mathrm{S}}$ and $\mathcal{P}_{\mathrm{R}}$ send $(\mathsf{Query}, \mathsf{sid}, m \| (c_{i,j})_{i\in[N]}^{j\in[n_{\mathcal{C}}]}, [N]^{n_{\mathcal{C}}})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and denote the response as $\Delta$. Each party then outputs $(\mathsf{Done}, \mathsf{sid}, \Delta)$.

**Deliver** On input $(\mathsf{Deliver}, \mathsf{sid})$, $\mathcal{P}_{\mathrm{S}}$ does as follows:

1. Broadcast $\mathbf{t}_{i,j}, r_{i,j}$ for $i \in [N] \setminus \{\Delta[i]\}, j \in [n_{\mathcal{C}}]$.

Upon observing the broadcast message $\mathbf{t}_{i,j}, r_{i,j}$ for $i \in [N] \setminus \{\Delta[i]\}, j \in [n_{\mathcal{C}}]$ from $\mathcal{P}_{\mathrm{S}}$ and if $\ell, H, m, (c_{i,j})_{i\in[N]}^{j\in[n_{\mathcal{C}}]}$ is broadcast by $\mathcal{P}_{\mathrm{S}}$ as well, $\mathcal{P}_{\mathrm{R}}$ acts as follows:

1. Send $(\mathsf{Query}, \mathsf{sid}, \mathbf{t}_{i,j} \| r_{i,j}, \{0,1\}^{2\lambda})$ to $\mathcal{F}_{\mathsf{GpRO}}$ for all $i \in [N] \setminus \{\Delta[j]\}, j \in [n_{\mathcal{C}}]$ and check that the response is $c_{i,j}$. If any $c_{i,j}$ is not correct, abort.
2. Compute $\mathbf{q}_j := \sum_{i\in[N]\setminus\{\Delta[j]\}} \mathbf{t}_{i,j}(\Delta[j] - i)$ for each $j \in [n_{\mathcal{C}}]$.
3. Set $\mathbf{Q} = [\mathbf{q}_1 \| \ldots \| \mathbf{q}_{n_{\mathcal{C}}}]$ and output $(\mathsf{Corr}, \mathsf{sid}, \mathbf{Q})$.

**Audit** On input $(\mathsf{Audit}, \mathsf{sid})$ act as follows:

1. Check that the messages $\ell, H, m, (c_{i,j})_{i\in[N]}^{j\in[n_{\mathcal{C}}]}$ and $\mathbf{t}_{i,j}, r_{i,j}$ for $i \in [N] \setminus \{\Delta[i]\}, j \in [n_{\mathcal{C}}]$ were broadcast in this order. If not, then abort.
2. Send $(\mathsf{Query}, \mathsf{sid}, \ell \| H, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and denote the response as $h$.
3. Send $(\mathsf{Query}, \mathsf{sid}, \ell \| (c_{i,j})_{i\in[N]}^{j\in[n_{\mathcal{C}}]}, \{0,1\}^{2\lambda})$ to $\mathcal{F}_{\mathsf{GpRO}}$. If the result differs from $H$ then abort.
4. Send $(\mathsf{Query}, \mathsf{sid}, m \| (c_{i,j})_{i\in[N]}^{j\in[n_{\mathcal{C}}]}, [N]^{n_{\mathcal{C}}})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain $\Delta$.
5. Compute $\mathbf{Q}$ as $\mathcal{P}_{\mathrm{R}}$ in **Deliver** and output $(\ell, h, m, \Delta, \mathbf{Q})$. Abort if it aborts.

**Challenge** If $\ell, H$ has been broadcast send $(\mathsf{Query}, \mathsf{sid}, \ell \| H, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$, obtain $h \in \mathcal{T}$ and output $(\mathsf{Challenge}, \mathsf{sid}, h)$. Otherwise output $(\mathsf{Challenge}, \mathsf{sid}, \perp)$.

Fig. 18: Protocol $\Pi_{\mathsf{IndWVole}}$

.

**Theorem 5.** *The protocol $\Pi_{\mathsf{IndWVole}}$, given in Figure 18, securely realizes $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ using the global subroutines $\mathcal{F}_{\mathsf{GpRO}}, \mathcal{F}_{\mathsf{GBB}}$. Any malicious adversary making $Q$ calls to $\mathcal{F}_{\mathsf{GpRO}}$ can distinguish with probability at most $Q/2^\lambda + Q/N^{n_\mathcal{C}} + Q^2/2^{2\lambda}$.*

As can be seen from the protocol $\Pi_{\mathsf{IndWVole}}$, all columns of $\mathbf{U}$ are independent and so are all indices of $\Delta$ chosen. Thus, we only give the security proof for $n_\mathcal{C} = 1$ as the generalization is obvious.

The following equation demonstrates why the protocol is indeed correct:

$$\mathbf{q} = \sum_{i \in [N] \setminus \{\Delta\}} \mathbf{t}_i(\Delta - i) = \sum_{i \in [N]} \mathbf{t}_i(\Delta - i) = \sum_{i \in [N]} \mathbf{t}_i \Delta - \sum_{i \in [N]} i \cdot \mathbf{t}_i = \mathbf{u}\Delta + \mathbf{v} \quad (1)$$

*Proof.*
*Malicious $\mathcal{P}_R$.* The high level idea is that the adversary gets to see every $\mathbf{t}_i$, except for $\mathbf{t}_\Delta$, as well as all the $r_i$ except $r_\Delta$. However, unless the adversary can guess $r_\Delta$ it cannot distinguish and we can rely on $\mathbf{t}_\Delta$ being random making $\mathbf{u} = \sum_{i \in [N]} \mathbf{t}_i$ uniformly random in $\mathbb{F}_p$.

We now construct a simulator. The simulator, upon receiving $(\mathsf{Init}, \mathsf{sid}, \ell, h)$ from $\mathcal{F}_{\mathsf{WVOLE}}$, will generate all $c_i$ as uniformly random commitments and program $\mathcal{F}_{\mathsf{GpRO}}$ to output the uniformly random $H \in \{0,1\}^{2\lambda}$ on input $\ell \| (c_i)_{i \in [N]}$. It also programs the random oracle to output $h$ on input $H$. Then it broadcasts $(\ell, H)$. Upon receiving $(\mathsf{Done}, \mathsf{sid}, m, \Delta)$ it programs $\mathcal{F}_{\mathsf{GpRO}}$ to output $\Delta$ on input $m \| (c_i)_{i \in [N]}$ and broadcasts $(m, (c_i)_{i \in [N]})$.

Upon receiving $(\mathsf{Corr}, \mathsf{sid}, \mathbf{q})$ it samples all $\mathbf{t}_i, i \neq \Delta$ uniformly at random from $\mathbb{F}_p^\ell$ subject to the constraint that they add up to $\mathbf{q}$ for $\Delta$ according to Equation 1. Then program $\mathcal{F}_{\mathsf{GpRO}}$ to output $c_i$ on input $\mathbf{t}_i \| r_i$ for uniformly random $r_i$, and broadcast all $\mathbf{t}_i, r_i$ for $i \neq \Delta$. The resulting protocol output is indistinguishable to an adversary as the $\mathbf{t}_i$ are chosen uniformly in both cases, unless it has sent a query to $\mathcal{F}_{\mathsf{GpRO}}$ before on input $\mathbf{t}_i, r_i$ so that it cannot be programmed to output $c_i$. Similarly, since the $c_i$ are chosen at random and the random oracle is programmed to output $\Delta$ before they are broadcast, $\mathcal{A}$ can only detect the programming if it queried the $\Delta$-generating input in advance. The same reasoning also holds for $H$. Since $r_i, c_i, H$ are chosen uniformly at random from $\{0,1\}^{2\lambda}$ by the simulator and the adversary makes at most $Q$ queries, this happens with probability at most $Q/2^{2\lambda}$.

*Malicious $\mathcal{P}_S$.* The simulator will react to queries to $\mathcal{F}_{\mathsf{GpRO}}$. For all inputs to $\mathcal{F}_{\mathsf{GpRO}}$, if they are of the form $\ell \| H$ where $H$ itself is a previous output of $\mathcal{F}_{\mathsf{GpRO}}$ on inputs $(c_i)_{i \in [N]}$ then find all $\mathcal{F}_{\mathsf{GpRO}}$ preimages of all $c_i$ (abort if there is more than one or one, and abort if there is more than one preimage for $H$). Construct

$\mathbf{u}, \mathbf{v}$ as in Equation 1, and use **Challenge Test** to generate the response $h$ to be output by $\mathcal{F}_{\mathsf{GpRO}}$. For repeated queries to $\mathcal{F}_{\mathsf{GpRO}}$, respond with the previous response. If some of the $(c_i)$ do not have preimages then emulate the RO honestly. In addition, for all inputs of the form $m\|(c_i)_{i\in[N]}$ when all $c_i$ are queried and valid as mentioned above, use **Delta Test** to generate the response $\Delta$ to be output by $\mathcal{F}_{\mathsf{GpRO}}$. For any other input to the RO, emulate the Random Oracle honestly.

The simulator, upon obtaining the $\ell\|H$ broadcast by the adversary, observes the random oracle queries that output $c_i$ as outlined above. Again, if there is any collision (i.e. two inputs generate the same $c_i$ as output) then abort. If any $c_i$ has no valid preimage, then sample the corresponding $\mathbf{t}_i$ uniformly at random. If the resulting query to $\mathcal{F}_{\mathsf{GpRO}}$ has not been made before that would generate $h$, then execute it as outlined in the previous paragraph. Construct $\mathbf{u}, \mathbf{v}$ as in Equation 1 and send them to $\mathcal{F}_{\mathsf{WVOLE}}$ where $c$ is chosen to be the consistent index with the previously made $\mathcal{F}_{\mathsf{GpRO}}$ query and $\mathcal{F}_{\mathsf{WVOLE}}$ response.

Wait for the adversary to broadcast $m, (c_i)_{i\in[N]}$ and check if $\ell\|(c_i)_{i\in[N]}$ hashes to $H$. If not then abort, otherwise send $(\mathsf{InitDone}, \mathsf{sid}, m)$ to $\mathcal{F}_{\mathsf{WVOLE}}$.

Wait for the adversary to broadcast openings according to all the $c_i$ except $c_\Delta$, as in the protocol. If any opening is invalid then abort. If $c_\Delta$ did not have a valid preimage as a random oracle output, then also abort. Otherwise, send $(\mathsf{Deliver}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{WVOLE}}$.

If all $c_i \in \{0,1\}^{2\lambda}$ have a unique preimage which are opened correctly by the adversary, then the simulation is perfect. If the adversary finds a collision in the Random Oracle for an output $c_i$ or for $H$ then the simulation aborts, but the probability for this event is $Q^2/2^{2\lambda}$ by Proposition 1 as the adversary makes at most $Q$ queries to $\mathcal{F}_{\mathsf{GpRO}}$. The simulation also aborts if more than one $c_i$ didn't have a preimage that could be extracted or if $H$ did not have a preimage, but in that case the adversary will not be able to open these during the **Deliver** phase of the protocol[2]. However, if the adversary did not generate $c_\Delta$ as a random oracle output but all other are correctly formed, then the simulation fails as the simulator aborts. However, $\Delta$ is chosen by the functionality at random after all $c_i$ are fixed via $\mathcal{F}_{\mathsf{GpRO}}$ in the simulation. Here, the adversary may query $\mathcal{F}_{\mathsf{GpRO}}$ (and thus trigger **Delta Test**) before broadcasting and the probability of $\mathcal{A}$ guessing $\Delta$ correctly is $1/N$ for each such query. If there are $n_{\mathcal{C}}$ parallel instances, then the chance of this event is $1/N^{n_{\mathcal{C}}}$. As the adversary can make at most $Q$ queries to the random oracle, his success probability of achieving this event is $Q/N^{n_{\mathcal{C}}}$ as stated in the theorem.

*Malicious $\mathcal{P}_S$ and $\mathcal{P}_R$.* The simulator works identical to the case of the dishonest sender outlined above.

[2] To be precise, the adversary could make random oracle queries after broadcasting $H$ or $m, c_i$ to match these with valid openings, but the success probability is bounded by $(N+1)Q/2^{2\lambda}$ which is substantially smaller than the $Q^2/2^{2\lambda}$ collision probability.

### D.2 From $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$

Let $\mathcal{C}$ be a systematic $[n_\mathcal{C}, d_\mathcal{C}, k_\mathcal{C}]$-code. We assume that it is linear, systematic and has generator matrix $\mathbf{G} \in \mathbb{F}_p^{k_\mathcal{C} \times n_\mathcal{C}}$.

In the construction, we use the $1/p$-universal hash functions as defined in Definition 2. Its matrices $\mathbf{H} \in \mathbb{F}_p^{1\times(n+1)}$ are $n$-hiding according to Definition 2.

The protocol follows a straightforward approach. $\mathcal{P}_\mathrm{S}$, upon obtaining $(\overline{\mathbf{U}}, \mathbf{V})$ from $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ considers the first $k_\mathcal{C}$ columns of $\overline{\mathbf{U}}$ as the output, whereas $\mathbf{V}$ remains untouched. Since we assume that $\mathcal{C}$ is systematic, we therefore have to correct the remaining $n_\mathcal{C} - k_\mathcal{C}$ columns of $\overline{\mathbf{U}}$ to be $\mathbf{UG}$. $\mathcal{P}_\mathrm{S}$ does this by computing the difference $[\mathbf{0}\,\mathbf{C}] = \mathbf{UG} - \overline{\mathbf{U}}$ and broadcasting $\mathbf{C}$. Note that the first $k_\mathcal{C}$ columns of $\mathbf{UG} - \overline{\mathbf{U}}$ must be zero, as they define the messages of the row codewords.

After receiving $\Delta, \overline{\mathbf{Q}}$ from $\mathcal{F}_{\mathsf{WVOLE}}$, $\mathcal{P}_\mathrm{S}$ can now correct it to $\mathbf{Q}$ by adding $[\mathbf{0}\,\mathbf{C}]\mathsf{diag}(\Delta)$ to it. This is correct because

$$\begin{aligned}\overline{\mathbf{Q}} + [\mathbf{0}\,\mathbf{C}]\mathsf{diag}(\Delta) &= \mathbf{V} + \overline{\mathbf{U}}\mathsf{diag}(\Delta) + [\mathbf{0}\,\mathbf{C}]\mathsf{diag}(\Delta)\\ &= \mathbf{V} + \overline{\mathbf{U}}\mathsf{diag}(\Delta) + (\mathbf{UG} - \overline{\mathbf{U}})\mathsf{diag}(\Delta)\\ &= \mathbf{V} + \mathbf{UG}\mathsf{diag}(\Delta)\end{aligned}$$

However, this correction does not make sure that $\mathbf{C}$ is correctly formed. Thus in addition, $\mathcal{P}_\mathrm{S}$ also samples an $\ell$-hiding universal linear hash function $\mathbf{H}$ based on $\mathbf{C}$ and applies it both to $\mathbf{U}$ and $\mathbf{V}$. It then broadcasts the resulting $\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}$. Looking ahead, if $\mathbf{Q} = \mathbf{UG}\mathsf{diag}(\Delta) + \mathbf{V}$, then $\mathbf{HQ} = \widetilde{\mathbf{U}}\mathbf{G}\mathsf{diag}(\Delta) + \widetilde{\mathbf{V}}$. At the same time, if $\mathbf{C}$ will not correct $\mathbf{Q}$ to the aforementioned relation (or if $\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}$ are not well-formed), then the resulting check including $\Delta$ will fail with very high probability over the random choice of $\Delta$. To produce $\ell$ output rows, we let the sender produce $\ell + 1$ correlated rows and use that $\mathbf{H}$ is $\ell$-hiding as it adds the last row in the process, which is discarded after the check.

Towards generating challenges, we assume that the hybrid functionality $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ in our protocol uses an output challenge set $\mathcal{T} = \{0,1\}^{2\lambda}$ and itself generates the same challenge set. We then hash the challenge it generates, together with the protocol transcript, to define the challenge output of this functionality.

**Theorem 6.** *Let $\mathcal{C}$ be a systematic $[n_\mathcal{C}, d_\mathcal{C}, k_\mathcal{C}]$-linear code. The protocol $\Pi_{\mathsf{CodeWVole}}$, given in Figure 19, securely realizes $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ in the $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$-hybrid model using the global subroutines $\mathcal{F}_{\mathsf{GpRO}}, \mathcal{F}_{\mathsf{GBB}}$. Any malicious adversary making $Q$ calls to $\mathcal{F}_{\mathsf{GpRO}}$ or **Challenge Test** or **Delta Test** can distinguish with probability at most $Q/p^{n_\mathcal{C}+k_\mathcal{C}} + Q\binom{n_\mathcal{C}}{k_\mathcal{C}+1}/p + Q^2/2^{2\lambda}$.*

If all parties are honest, then completeness is clear from the description above. Our proof generally follows the strategy in [7].

**Protocol $\Pi_{\mathsf{CodeWVole}}$**

$\mathcal{H} \subseteq \mathbb{F}_p^{1\times(\ell+1)}$ is a family of $\ell$-hiding, $1/p$-universal linear hash functions. The protocol uses $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}, \mathcal{F}_{\mathsf{GpRO}}, \mathcal{F}_{\mathsf{GBB}}$ as hybrid functionalities. Let $\mathcal{T} = \{0,1\}^{2\lambda}$ be the challenge space.

**Init** On $(\mathsf{Init}, \mathsf{sid}, \ell)$, $\mathcal{P}_\mathrm{S}$ runs the following protocol:

1. Send $(\mathsf{Init}, \mathsf{sid}, \ell+1)$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$.
2. Receive $\overline{\mathbf{U}} \in \mathbb{F}_p^{(\ell+1)\times n_\mathcal{C}}$, $\mathbf{V} \in \mathbb{F}_p^{(\ell+1)\times n_\mathcal{C}}$ and $h$ from $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$.
3. Let $\mathbf{U} = \overline{\mathbf{U}}^{[1..k_\mathcal{C}]}$ be the first $k_\mathcal{C}$ columns of $\overline{\mathbf{U}}$.
4. Let $[\mathbf{0}\,\mathbf{C}] := \mathbf{UG} - \overline{\mathbf{U}}$ where $\mathbf{C} \in \mathbb{F}_p^{(\ell+1)\times(n_\mathcal{C}-k_\mathcal{C})}$ and broadcast $\mathbf{C}$.
5. Choose $\mathbf{H} \in \mathcal{H}$ as the response to the query $(\mathsf{Query}, \mathsf{sid}, h\|\mathbf{C}, \mathbb{F}_p^\ell)$ to $\mathcal{F}_{\mathsf{GpRO}}$ and compute $\widetilde{\mathbf{U}} := \mathbf{HU}$ and $\widetilde{\mathbf{V}} := \mathbf{HV}$.
6. Broadcast $(\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}})$ and output $(\mathbf{U}_{[1..\ell]}, \mathbf{V}_{[1..\ell]})$.

$\mathcal{P}_\mathrm{R}$, upon obtaining $(\mathsf{Init}, \mathsf{sid}, \ell, h)$ from $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ and observing the broadcasts $\mathbf{C}, \widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}$ queries $\mathcal{F}_{\mathsf{GpRO}}$ on input $(\mathsf{Query}, \mathsf{sid}, h\|\mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}, \mathcal{T})$ to obtain $h'$. Then it outputs $(\mathsf{Done}, \mathsf{sid}, \ell, h')$.

**Init Done** On input $(\mathsf{InitDone}, \mathsf{sid}, m)$ $\mathcal{P}_\mathrm{S}$ sends $(\mathsf{InitDone}, \mathsf{sid}, \mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}\|m)$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$. Both $\mathcal{P}_\mathrm{S}$ and $\mathcal{P}_\mathrm{R}$ obtain $m, \Delta$, and $\mathcal{P}_\mathrm{R}$ checks that the output of $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ is consistent with the broadcast $\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}, \mathbf{C}$.

**Deliver** On $(\mathsf{Deliver}, \mathsf{sid})$, $\mathcal{P}_\mathrm{S}$ does the following:

1. Send $(\mathsf{Deliver}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$.

$\mathcal{P}_\mathrm{R}$ then does the following:

1. Check if $\mathcal{P}_\mathrm{S}$ broadcast $\mathbf{C}$ first and then $(\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}})$. If not, abort.
2. Obtain $(\mathsf{Corr}, \mathsf{sid}, \overline{\mathbf{Q}})$ from $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$. By definition, $\overline{\mathbf{Q}} = \overline{\mathbf{U}}\mathsf{diag}(\Delta) + \mathbf{V}$.
3. Compute $\mathbf{Q} := \overline{\mathbf{Q}} + [\mathbf{0}\,\mathbf{C}]\mathsf{diag}(\Delta)$.
4. Recompute $\mathbf{H}$ as the response to sending $(\mathsf{Query}, \mathsf{sid}, h\|\mathbf{C}, \mathbb{F}_p^\ell)$ to $\mathcal{F}_{\mathsf{GpRO}}$.
5. Abort if $\widetilde{\mathbf{V}} \neq \mathbf{HQ} - \widetilde{\mathbf{U}}\mathbf{G}\mathsf{diag}(\Delta)$. Otherwise output $(\mathbf{Q}_{[1..\ell]})$.

**Audit** On input $(\mathsf{Audit}, \mathsf{sid})$ $\mathcal{P}_\mathrm{A}$ acts as follows:

1. Obtain $(\ell, h, \mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}\|m, \Delta, \overline{\mathbf{Q}})$ by sending $(\mathsf{Audit}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$. If there are no broadcasts of $\mathbf{C}$ or $\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}$ on $\mathcal{F}_{\mathsf{GBB}}$ then abort.
2. Send $(\mathsf{Query}, \mathsf{sid}, h\|\mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ to obtain $h'$.
3. Compute $\mathbf{Q}$ as $\mathcal{P}_\mathrm{R}$ does in **Deliver** and abort if it aborts.
4. Output $(\ell, h', m, \Delta, \mathbf{Q})$

**Challenge** On input $(\mathsf{Challenge}, \mathsf{sid})$:

- Send $(\mathsf{Challenge}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ and obtain $h$. If $h = \perp$ or $\mathbf{C}, \widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}$ have not been broadcast then output $(\mathsf{Challenge}, \mathsf{sid}, \perp)$.
- Query $\mathcal{F}_{\mathsf{GpRO}}$ on input $(\mathsf{Query}, \mathsf{sid}, h\|\mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}, \mathcal{T})$ to obtain $h'$. Then output $(\mathsf{Challenge}, \mathsf{sid}, h')$.

Fig. 19: Protocol $\Pi_{\mathsf{CodeWVole}}$ achieving subspace VOLE

*Proof.*

*Malicious $\mathcal{P}_R$.* The simulator works as follows: it honestly simulates an instance

of $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ towards the adversary. Upon obtaining $(\mathsf{Done}, \mathsf{sid}, \ell, h)$ from $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ it first samples $\overline{h} \in \{0,1\}^{2\lambda}$ uniformly at random. Then sample $\mathbf{C} \in \mathbb{F}_p^{(\ell+1)\times(n_\mathcal{C}-k_\mathcal{C})}$ uniformly at random. Choose $\mathbf{H}$ consistently with $\mathcal{F}_{\mathsf{GpRO}}$ and choose uniformly random $\widetilde{\mathbf{U}} \in \mathbb{F}_p^{k_\mathcal{C}}, \widetilde{\mathbf{V}} \in \mathbb{F}_p^{n_\mathcal{C}}$. Program $\mathcal{F}_{\mathsf{GpRO}}$ to output $h$ on input $\overline{h}\|\mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}$ where $\overline{h}$ was chosen as outlined above. Finally, we let $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ output $(\mathsf{Init}, \mathsf{sid}, \ell+1, \overline{h})$ and broadcast $\mathbf{C}$ as well as $(\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}})$.

Upon obtaining $(\mathsf{InitDone}, \mathsf{sid}, m, \Delta)$ from $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ let $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ deliver $(\mathsf{Done}, \mathsf{sid}, \mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}\|m, \Delta)$.

Upon obtaining $(\mathsf{Corr}, \mathsf{sid}, \mathbf{Q})$ from $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ set $\overline{\mathbf{Q}}_{[1..\ell]} = \mathbf{Q} - [\mathbf{0}\,\mathbf{C}]_{[1..\ell]}\mathsf{diag}(\Delta)$ and $\overline{\mathbf{Q}}_{\ell+1} = \widetilde{\mathbf{V}} + \widetilde{\mathbf{U}}\mathbf{G}\mathsf{diag}(\Delta) - \mathbf{H}^{[1..\ell]}\mathbf{Q}_{[1..\ell]}$. Let $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ output $(\mathsf{Corr}, \mathsf{sid}, \Delta, \overline{\mathbf{Q}})$.

Note that by construction, the chosen $\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}, \mathbf{C}$ and the $\Delta, \widetilde{\mathbf{Q}}$ fulfill the check in Step 5 of **Deliver** of $\Pi_{\mathsf{CodeWVole}}$. It remains to show that all other values have the same distribution as in the protocol.

Note that $\mathbf{C}$ as generated in the protocol includes the term $\overline{\mathbf{U}}$ which is uniformly random. $\mathcal{P}_\mathrm{R}$ never learns $\overline{\mathbf{U}}$, but only $\overline{\mathbf{Q}}$ which additionally contains the uniformly random additive term $\mathbf{V}$. Thus, $\mathbf{C}$ is identically distributed in both cases. Moreover, by the $\ell$-hiding property of the universal hash function family $\mathcal{H}$ and the fact that $\mathbf{U}_{\ell+1}, \mathbf{V}_{\ell+1}$ are uniformly random in $\mathbb{F}_p^{k_\mathcal{C}}$ and $\mathbb{F}_p^{n_\mathcal{C}}$ respectively, $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ also have the same distribution in both settings. Finally, the value $\Delta$ is consistent. The rows $\overline{\mathbf{Q}}$ are uniquely defined to be consistent with $\mathbf{U}, \mathbf{V}, \Delta$ of $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ and the last row is uniquely defined by the check in Step 5. Thus, the simulation is perfectly indistinguishable except for calls of $\mathcal{A}$ to $\mathcal{F}_{\mathsf{GpRO}}$. However, when we program $\mathcal{F}_{\mathsf{GpRO}}$ on input $\overline{h}\|\mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}$ to output $h$ then $\mathcal{A}$ had to predict the uniformly random input $\overline{h} \in \{0,1\}^{2\lambda}$ in order to detect the programming. Given that $\mathcal{A}$ makes $Q$ queries to $\mathcal{F}_{\mathsf{GpRO}}$, this happens with probability at most $Q/2^{2\lambda}$.

*Malicious $\mathcal{P}_S$.* The simulator again honestly simulates an instance of $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ towards the adversary. We will take care of the adversarial calls to **Challenge Test** and **Delta Test** below.

The hybrid functionality outputs $\overline{\mathbf{U}}, \mathbf{V}$ to the adversary (or lets it choose both), who in response broadcasts $\mathbf{C}$ and thereafter $\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}$. Note that this is tied to one choice of $h$ generated by $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$.

*Generate candidate correlation inputs.* From these broadcasts, the simulator must first obtain inputs for the ideal functionality $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ which are consistent with the protocol messages, and send them to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ for **Init**.

Recompute $\mathbf{H}$ by making the query $(\mathsf{Query}, \mathsf{sid}, h \| \mathbf{C}, \mathbb{F}_p^\ell)$ to $\mathcal{F}_{\mathsf{GpRO}}$ as in the protocol and set $\mathbf{U}_D := \overline{\mathbf{U}} + [\mathbf{0}\,\mathbf{C}]$. Then

$$\begin{aligned}\mathbf{Q} &= \overline{\mathbf{Q}} + [\mathbf{0}\,\mathbf{C}]\mathsf{diag}(\Delta) \\ &= \overline{\mathbf{U}}\mathsf{diag}(\Delta) + \mathbf{V} + [\mathbf{0}\,\mathbf{C}]\mathsf{diag}(\Delta) \\ &= \mathbf{U}_D\mathsf{diag}(\Delta) + \mathbf{V}\end{aligned}$$

would be obtained by the receiver. The adversary is supposed to respond with $\mathbf{H}\mathbf{U}_D$ and $\mathbf{H}\mathbf{V}$ in the consistency check. We define the differences to the actual values $\widetilde{\mathbf{U}}$, $\widetilde{\mathbf{V}}$ which it sends as $\widehat{\mathbf{U}} = \mathbf{H}\mathbf{U}_D - \widetilde{\mathbf{U}}\mathbf{G}$ and $\widehat{\mathbf{V}} = \mathbf{H}\mathbf{V} - \widetilde{\mathbf{V}}$.

For the correct responses, $\mathbf{H}\mathbf{V} = \mathbf{H}\mathbf{Q} - \mathbf{H}\mathbf{U}_D\mathsf{diag}(\Delta)$ must hold for all $\Delta$. If the protocol consistency check $\widetilde{\mathbf{V}} = \mathbf{H}\mathbf{Q} - \widetilde{\mathbf{U}}\mathbf{G}\mathsf{diag}(\Delta)$ also holds, then we have that $\widehat{\mathbf{V}} = -\widehat{\mathbf{U}}\mathsf{diag}(\Delta)$. Thus, for the check to pass, we can solve this system of linear equations for $\Delta$.

If no solution for $\Delta$ exists, then the simulator aborts. This is because the adversary cannot pass the check for any choice of $\Delta$ of the ideal functionality. Otherwise, call such a solution $\Delta^*$. There are indices $\mathtt{Cols} \subseteq [n_{\mathcal{C}}]$ corresponding to the non-zero columns of $\widehat{\mathbf{U}}$ and for those $i \in \mathtt{Cols}$ we must later have that $\Delta[i] = \Delta^*[i]$. Otherwise, the protocol consistency check will fail anyways.

Let $\mathcal{C}_{-\mathtt{Cols}}$ be the code $\mathcal{C}$ punctured at all indices in $\mathtt{Cols}$. We decode all rows of $\mathbf{U}_D$ after removing columns corresponding to $\mathtt{Cols}$ according to $\mathcal{C}_{-\mathtt{Cols}}$. If any row of $\mathbf{U}_D$ (punctured at $\mathtt{Cols}$) is not already in $\mathcal{C}_{-\mathtt{Cols}}$ then abort. If $|\mathtt{Cols}| > d_{\mathcal{C}}$ then pick a random decoding that is consistent. Otherwise, let $\mathbf{U}^*$ be the decoded rows, set $\mathbf{V}^* = \mathbf{V} + (\mathbf{U}_D - \mathbf{U}^*\mathbf{G})\mathsf{diag}(\Delta^*)$ and send the first $\ell$ rows of $\mathbf{U}^*, \mathbf{V}^*$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$. This means that the simulator will first send $(\mathsf{TestChallenge}, \mathsf{sid}, \ell, \mathbf{U}^*_{[1..\ell]}, \mathbf{V}^*_{[1..\ell]})$ and then $(\mathsf{Sender}, \mathsf{sid}, \mathbf{U}^*_{[1..\ell]}, \mathbf{V}^*_{[1..\ell]}, 0)$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$.

*Correctness of the extracted inputs.* If the functionality $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ delivers output, then upon fixing $\Delta$ it holds that

$$\begin{aligned}\mathbf{V}^* &= \mathbf{V} + \mathbf{U}_D\mathsf{diag}(\Delta) - \mathbf{U}^*\mathbf{G}\mathsf{diag}(\Delta) \\ &= \mathbf{Q} - \mathbf{U}^*\mathbf{G}\mathsf{diag}(\Delta)\end{aligned}$$

This is consistent on all columns not in $\mathtt{Cols}$ as $\mathbf{U}_D - \mathbf{U}^*\mathbf{G}$ vanishes, and for those in $\mathtt{Cols}$ it is correct if $\Delta[i] = \Delta^*[i]$, which holds if the consistency check in the protocol passed. This also means that the outputs of $\mathcal{F}_{\mathsf{WVOLE}}$ as well as the protocol fulfill the same required correlation.

Note that we abort the simulation in two cases: either if no solution $\Delta^*$ could be found, or if some rows in the punctured $\mathbf{U}_D$ do not lie in $\mathcal{C}_{-\mathtt{Cols}}$. For the first event, this implies that the consistency check will fail and can be discarded. Therefore, we must bound the probability of the second event. For this, let $\mathcal{K}$ be the collection of sets of columns of $\mathbf{G}$ such that every $c \in \mathcal{K}$ is linearly dependent, while all proper subsets of $c$ are linearly independent. By construction, $|\mathcal{K}| =$

$\binom{n_\mathcal{C}}{\dim+1}$. Hence, every element in $\mathcal{K}$ is a proper sub-code of $\mathcal{C}$ with minimal distance 1.

If the consistency check of the protocol passes, then

$$\widetilde{\mathbf{V}} = \mathbf{HQ} - \widetilde{\mathbf{U}}\mathbf{G}\mathsf{diag}(\Delta) = \mathbf{HV} + \mathbf{HU}_D\mathsf{diag}(\Delta) - \widetilde{\mathbf{U}}\mathbf{G}\mathsf{diag}(\Delta).$$

This requires that $\mathbf{HU}_D$ is in $\mathcal{C}$ due to the free choice of $\Delta$. Therefore, the indices of $\mathbf{HU}_D$ that are not in $\mathtt{Cols}$ must be in the punctured code $\mathcal{C}_{-\mathtt{Cols}}$. We now bound the probability that one (or more rows) of $\mathbf{U}_D$ punctured at $\mathtt{Cols}$ are not in $\mathcal{C}_{-\mathtt{Cols}}$, while $\mathbf{HU}_D$ punctured at $\mathtt{Cols}$ is in $\mathcal{C}_{-\mathtt{Cols}}$.

Towards this, let $\mathbf{P}_\mathcal{C}$ be the parity check matrix of the code $\mathcal{C}$. $\mathbf{HU}_D$ is in $\mathcal{C}$ if and only if $\mathbf{HU}_D\mathbf{P}_\mathcal{C} = \mathbf{0}$. Note that every $c \in \mathcal{K}$ induces a code $\mathcal{C}_c$ of length $k_\mathcal{C}+1$ and dimension $k_\mathcal{C}$. Thus, for every $c \in \mathcal{K}$ we can define a parity check vector $\mathbf{p}_c \in \mathbb{F}_p^{k_\mathcal{C}+1}$ for $\mathcal{C}_c$ so that $\forall \mathbf{x} \in \mathbb{F}_p^{k_\mathcal{C}+1} : \mathbf{x} \in \mathcal{C}_c \leftrightarrow \langle \mathbf{x}, \mathbf{p}_c\rangle = 0$.

In general, for any $\mathbf{x} \in \mathbb{F}_p^{n_\mathcal{C}}$ we have that if for all $c \in \mathcal{K}$ it holds that[3] $\langle \mathbf{x}_c, \mathbf{p}_c\rangle = 0$, then it holds that $\mathbf{xP}_\mathcal{C} = \mathbf{0}$. Therefore, in order to bound the probability that a matrix $\mathbf{U}_D$ is not in $\mathcal{C}$ (or a submatrix not in the punctured code) while $\mathbf{HU}_D$ being in $\mathcal{C}$, we can look at the individual event for each code $\mathcal{C}_c$ and the chosen $\mathbf{H}$ and then apply a union bound over all $c \in \mathcal{K}$. Due to the uniform choice of $\mathbf{H}$ after $\mathbf{U}_D$ is fixed, the probability of $\forall c \in \mathcal{K} : \mathbf{H}(\mathbf{U}_c\mathbf{p}_c^\top) = \mathbf{0}$ can be bounded by $\binom{n_\mathcal{C}}{k_\mathcal{C}+1}/p$. As the adversary may make up to $Q$ queries to the Random Oracle using different choices of $\mathbf{C}$, each resulting in a different $\mathbf{H}$, its success probability is $Q\binom{n_\mathcal{C}}{k_\mathcal{C}+1}/p$.

*Identifying the correct inputs to the hybrid functionality.* The aforementioned argument requires that the simulator can identify the correct $\overline{\mathbf{U}}, \mathbf{V}$ for the input extraction, which is only possible if they were previously provided to the simulated $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ by $\mathcal{A}$. However, since the query that generates $\mathbf{H}$ also includes the input $h \in \mathcal{T}$ that is chosen uniformly at random by the simulator only after it obtained the inputs $\overline{\mathbf{U}}, \mathbf{V}$ from the adversary, the attacker must essentially find two inputs to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ which generate a collision on $h$ values. The probability of this event is at most $Q^2/2^{2\lambda}$ due to Proposition 1.

*Simulating $h'$.* In order to simulate the $\mathcal{F}_{\mathsf{GpRO}}$ queries that generate $h'$, note that if $\mathcal{A}$ has sent $\overline{\mathbf{U}}, \mathbf{V}$ corresponding to $\mathbf{C}, \widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ then it has obtained $h$, which was chosen uniformly at random. Thus, whenever we see an $\mathcal{F}_{\mathsf{GpRO}}$ input query with a $h$ that we generated for an input, we can perfectly simulate the query by computing $\mathbf{U}, \mathbf{V}$ for the **Challenge Test** interface (as well as $\ell$) of $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ as outlined in *Generate candidate correlation inputs* above. Given its output $h'$ we then program $\mathcal{F}_{\mathsf{GpRO}}$ to give the response $h'$ that was obtained from $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$. For all other $h$ that were not generated by $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ we respond with random outputs. However, this fails if the wrong input is extracted or if the

[3] Here, $\mathbf{x}_c$ is the subvector of $\mathbf{x}$ punctured at all indices except $c$.

adversary guesses a $h$ that is later generated by $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$. In the former case, this probability is at most $Q\binom{n_\mathcal{C}}{k_\mathcal{C}+1}/p$ over $Q$ queries to $\mathcal{F}_{\mathsf{GpRO}}$. For the latter event, this happens with probability at most $Q^2/2^{2\lambda}$ due to Proposition 1.

*Simulating **Challenge Test** and **Delta Test** queries.* For any **Challenge Test** query with input $\overline{\mathbf{U}}, \mathbf{V}$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ output a uniformly random value $h$. For any **Delta Test** query $(\ell+1, h, \mathbf{C}\|\widetilde{\mathbf{U}}\|\widetilde{\mathbf{V}}\|m)$ to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ where $h$ was output by $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ before, we can use the **Challenge Test** interface of $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ as in the $h'$ simulation case to compute the corresponding $h'$. We can then send $(\ell, h', m)$ to **Delta Test** of $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$ and obtain $\Delta$. Finally, we let $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ output this value $\Delta$. If $h$ was not generated by **Challenge Test** before then generate $h'$ as a random value and use it for the **Delta Test** query to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$. This fails if we extract the incorrect input to **Challenge Test** of $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathcal{C}}$, which happens for all $Q$ queries with probability at most $Q\binom{n_\mathcal{C}}{k_\mathcal{C}+1}/p$. Alternatively, this fails if different inputs were provided to $\mathcal{F}_{\mathsf{WVOLE}}^{p,N,\mathbb{F}_p^{n_\mathcal{C}}}$ which generate the same $h$. This event happens with probability at most $Q^2/2^{2\lambda}$ due to Proposition 1. In addition, we can have that we program the output of **Delta Test** for a previously unqueried input $h$ that is only later generated by **Challenge Test**. This event happens with probability at most $Q/2^{2\lambda}$.

*Malicious $\mathcal{P}_S$ and $\mathcal{P}_R$.* The proof is identical to the setting of the dishonest $\mathcal{P}_\mathrm{S}$ above, as $\mathcal{P}_\mathrm{R}$ does not do anything in the protocol and the output of $\mathcal{P}_\mathrm{A}$ is identical to that of an honest $\mathcal{P}_\mathrm{R}$.

## E Proof of Theorem 1 (Single receiver Commitments)

For the proof, we will provide three simulators: $\mathcal{S}_{\mathsf{HOT-1-HonestSender}}$ against a dishonest receiver, $\mathcal{S}_{\mathsf{HOT-1-HonestReceiver}}$ against a dishonest receiver and $\mathcal{S}_{\mathsf{HOT-1-BothCorrupted}}$ where both the sender and receiver can be dishonest. This is sufficient, as corruptions are static.

*Proof.*

*$\mathcal{S}_{\mathsf{HOT-1-HonestSender}}$.* The simulator simulates $\mathcal{F}_{\mathsf{WVOLE}}$ and has access to $\mathcal{F}_{\mathsf{GpRO}}, \mathcal{F}_{\mathsf{GBB}}$. Whenever $\mathcal{P}_\mathrm{R}$ broadcasts $(\mathsf{Abort}, \mathsf{sid})$, $\mathcal{S}_{\mathsf{HOT-1-HonestSender}}$ sends $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$.

**Init** Upon receiving message $(\mathsf{Init}, \mathsf{sid}, \mathrm{mc})$ from $\mathcal{F}_{\mathsf{HOTCom}}$ $\mathcal{S}_{\mathsf{HOT-1-HonestSender}}$ simulates the protocol and all functionalities honestly.

**Commit** Upon obtaining $(\mathsf{Receipt}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, \mathbf{ch})$ choose all $d_j \in \mathbb{F}_p$ uniformly at random and program $\mathcal{F}_{\mathsf{GpRO}}$ to output $\mathbf{ch}$ on input $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}}\|\mathtt{mp}_{\mathrm{com}})$. If the programming fails, then abort. Otherwise, run the protocol.

**Add and Linear** Follow the protocol honestly.

**Hint** Upon receiving $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j, x_j)_{j\in[k]}, 0)$ from $\mathcal{F}_{\mathsf{HOTCom}}$, broadcast $(\mathsf{cid}_j, x_j)_{j\in[k]}$ and otherwise follow the protocol. Upon receiving $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j, x_j)_{j\in[k]}, 1)$ from $\mathcal{F}_{\mathsf{HOTCom}}$, sample uniformly random $c_j \in \{0,1\}^{2\lambda}$. Then broadcast $(\mathsf{cid}_j, c_j)_{j\in[k]}$ and send $(\mathsf{cid}_j, x_j)_{j\in[k]}$ to $\mathcal{P}_{\mathrm{R}}$.

**Delayed Hint** Upon receiving $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$ from $\mathcal{F}_{\mathsf{HOTCom}}$, sample uniformly random $c_j \in \{0,1\}^{2\lambda}$. Then broadcast $(\mathsf{cid}_j, c_j)_{j\in[k]}$.

**Open** Upon receiving the first $(\mathsf{Open}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{HOTCom}}$, the simulator chooses $\Delta_1$ uniformly at random. It computes $\mathbf{S}, \mathbf{D}$ according to the protocol, computes $\Gamma$ as the verifier would compute it in the protocol and sets all $g_j$ for opened messages such that $g_j = f_j^{(0)} + f_j^{(1)}(\mathbf{s}'_1, \ldots, \mathbf{s}'_\ell) - m_j\Delta_1$, which is always uniquely possible by construction. Moreover, it samples $r_j$ for $B_{\mathtt{priv}}$ uniformly at random, which now fully defines $A_{\mathtt{pub}}, A_{\mathtt{priv}}, A_{\mathtt{delay}}, B_{\mathtt{priv}}$. The simulator reprograms $\mathcal{F}_{\mathsf{GpRO}}$ to output $\Delta_1$ on input $\overline{\mathbf{ch}}\|A_{\mathtt{pub}}\|A_{\mathtt{priv}}\|A_{\mathtt{delay}}$. If the adversary sent this input to $\mathcal{F}_{\mathsf{GpRO}}$ before, though, then the simulator aborts. The simulator also reprograms the random oracle to output $c_j$ on input $m_j\|g_j\|r_j$ for each $c_j$ generated during **Private Hint**. Again, if the adversary queried $m_j\|g_j\|r_j$ to $\mathcal{F}_{\mathsf{GpRO}}$ before then abort. Finally, it broadcasts $\mathbf{S}, \mathbf{D}, A_{\mathtt{pub}}, A_{\mathtt{delay}}$ and sends $B_{\mathtt{priv}}$ to $\mathcal{P}_{\mathrm{R}}$.

**Delayed Open** Upon receiving the second $(\mathsf{Open}, \mathsf{sid}, (\mathsf{cid}_j, m_j)_{j\in[k]})$ from $\mathcal{F}_{\mathsf{HOTCom}}$, the simulator computes the remaining $g_j = f_j^{(0)} + f_j^{(1)}(\mathbf{s}'_1, \ldots, \mathbf{s}'_\ell) - m_j\Delta_1$. It then samples $r_j$ for $A_{\mathtt{delay,open}}$ uniformly at random, and reprograms the Random Oracle to output $c_j$ on input $m_j\|g_j\|r_j$ for each $c_j$ generated during **Delayed Hint**. If the adversary queried $m_j\|g_j\|r_j$ to $\mathcal{F}_{\mathsf{GpRO}}$ before then abort. Finally, it broadcasts $A_{\mathtt{delay,open}}$.

**Verify, Audit and Challenges** Simulate $\mathcal{F}_{\mathsf{WVOLE}}$ honestly.

We now argue indistinguishability of transcripts via a series of hybrids.

Let $\mathcal{H}_0$ be the distribution of all the messages towards the adversary and honest outputs as in the protocol $\Pi^{1-1}_{\mathsf{HOTCommit}}$.

Let $\mathcal{H}_1$ be defined like $\mathcal{H}_0$, except that we now sample the commitments $c_j$ uniformly at random as in the simulation and only later program the random oracle so that the consistency checks pass. The commitments $c_j$ obtained during **Hint** and **Delayed Hint** are outputs of $\mathcal{F}_{\mathsf{GpRO}}$ in the protocol. As $\mathcal{F}_{\mathsf{GpRO}}$ is a random oracle, the $c_j$ are indeed indistinguishably distributed given that they are computed in the protocol using a uniformly random string $r_j \in \{0,1\}^\lambda$ only revealed later to $\mathcal{P}_{\mathrm{R}}$. Note that the simulator aborts if it fails to program the Random Oracle on inputs that lead to $c_j$. But to query on an input that leads to $c_j$, $\mathcal{A}$ had to guess a string $r_j$ of length $\lambda$ bits correctly. As the adversary makes $Q$ queries to the Random Oracle, his success probability in guessing $r_j$ (and thus aborting the protocol and distinguishing between $\mathcal{H}_0$ and $\mathcal{H}_1$) is at most $Q/2^\lambda$.

Let $\mathcal{H}_2$ be defined like $\mathcal{H}_1$, except that we choose $d_j$ obtained in **Commit** uniformly at random as in the simulation, but then change the $g_j$ according to $m_j$. We also now program $\mathcal{F}_{\mathsf{GpRO}}$ to output challenge values $h$ accordingly. This is perfectly indistinguishable, as $m_j$ is blinded in the protocol by an element from $\mathbf{U}_1$ that is chosen uniformly at random by $\mathcal{F}_{\mathsf{WVOLE}}$, but may fail as we program the random oracle. Note that the adversary never learns anything about $\mathbf{U}_1$ throughout the protocol except a sum involving $\mathbf{U}_2$ where $\mathbf{U}_2$ is also uniformly random. To address the abort probability, note that the adversary makes at most $Q$ queries to $\mathcal{F}_{\mathsf{GpRO}}$, where during each query it might have guessed $d_j$ correctly, so the error probability is $Q^2/p$.

Let $\mathcal{H}_3$ be defined like $\mathcal{H}_2$, except that we now also choose $\Delta_1$ uniformly at random as in the simulation and have the simulated $g_j$ fulfill the equation like the real $g_j$ do in the protocol. Note that $\mathbf{S}, \mathbf{D}$ are computed as in the protocol and therefore have the same distribution. Also, this sampling fixes $A_{\mathtt{pub}}, A_{\mathtt{delay}}$ and $B_{\mathtt{priv}}$ to have the same distribution in the protocol and in the simulation. $\mathcal{H}_3$ is identical to the distribution of the simulator.

Note that the simulator again aborts if it fails to program the Random Oracle on the input that leads to $\Delta_1$. If the last **Hint** before the call **Open** had $i = 1$, then due to the random choice of $c_j$ sent to $\mathcal{P}_{\mathrm{R}}$ the abort probability trivially follows by observing that the revealed $r_j$ is chosen uniformly at random during the simulation. If the last message was **Hint** with $i = 0$, then note that we send $A_{\mathtt{pub}}$ during **Open** that contains $g_j$ which is chosen uniformly at random, leading to the same abort probability of most $Q/2^\lambda$. Finally, if the last message was a **Delayed Hint** then we could have chosen $\Delta_1$ at that point where also a uniformly random $c_j \in \{0,1\}^{2\lambda}$ was generated by the simulator and that is part of the query that generates $\Delta_1$.

$\mathcal{S}_{\mathsf{HOT-1-HonestReceiver}}$. The simulator simulates $\mathcal{F}_{\mathsf{WVOLE}}$ and has access to $\mathcal{F}_{\mathsf{GpRO}}$ and $\mathcal{F}_{\mathsf{GBB}}$. Whenever $\mathcal{P}_{\mathrm{S}}$ broadcasts $(\mathsf{Abort}, \mathsf{sid})$, $\mathcal{S}_{\mathsf{HOT-1-HonestReceiver}}$ sends $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$. Note that at any point of the protocol, the adversary can send $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}} \| \dots)$ to $\mathcal{F}_{\mathsf{GpRO}}$ whose response we compute by looking up which query on $\mathcal{F}_{\mathsf{WVOLE}}$ generated the output $\overline{\mathbf{ch}}$ and then translating the other inputs of the RO query into a **Challenge Test** query at $\mathcal{F}_{\mathsf{HOTCom}}$. The output of this query will become the response of $\mathcal{F}_{\mathsf{GpRO}}$, i.e. we program the random oracle on this point. The programming only fails if the adversary ever predicts $\overline{\mathbf{ch}}$ from $\mathcal{F}_{\mathsf{WVOLE}}$ and makes a random oracle query that we respond to without querying $\mathcal{F}_{\mathsf{HOTCom}}$, which only happens with probability at most $Q^2/2^{2\lambda}$ by Proposition 1. We now proceed with the protocol simulation:

**Init** $\mathcal{S}_{\mathsf{HOT-1-HonestReceiver}}$ simulates the protocol and $\mathcal{F}_{\mathsf{WVOLE}}$ honestly. If $\mathcal{P}_{\mathrm{S}}$ sent the correct $(\mathsf{Init}, \mathsf{sid}, 2\ell)$ to $\mathcal{F}_{\mathsf{WVOLE}}$ then calculate mc and send $(\mathsf{Init}, \mathsf{sid}, \mathrm{mc})$ in its name to $\mathcal{F}_{\mathsf{HOTCom}}$. Store the matrices $\mathbf{U}, \mathbf{V}$ that $\mathcal{P}_{\mathrm{S}}$ obtained from (or supplied to) $\mathcal{F}_{\mathsf{WVOLE}}$ and the challenge $\overline{\mathbf{ch}}$ generated by it.

**Commit** Upon obtaining all $d_j$ via broadcast from $\mathcal{P}_{\mathrm{S}}$, recompute $m_j = d_j + \mathbf{U}[x+1, y+1]$ for $x, y$ as computed in the protocol based on the stored

matrix $\mathbf{U}$ for $\overline{\mathbf{ch}}$. Then send $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{cid}_{\iota_j}, m_j)_{j\in[k]})$ to $\mathcal{F}_{\mathsf{HOTCom}}$ and otherwise follow the protocol.

**Add and Linear** Follow the protocol honestly.

**Hint** Upon receiving the broadcast $(\mathsf{cid}_j, x_j)_{j\in[k]}$ from $\mathcal{P}_{\mathrm{S}}$ send $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j, x_j)_{j\in[k]}, 0)$ to $\mathcal{F}_{\mathsf{HOTCom}}$ and otherwise follow the protocol. If any $x_j$ is different from the $m_j$ $\mathcal{P}_{\mathrm{S}}$ should have sent based on the values extracted during **Commit** then set the flag $\texttt{cheated1} \leftarrow \top$.

Upon receiving the broadcast $(\mathsf{cid}_j, c_j)_{j\in[k]}$ and the private message $(\mathsf{cid}_j, x_j)$ from $\mathcal{P}_{\mathrm{S}}$ send $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j, x_j)_{j\in[k]}, 1)$ to $\mathcal{F}_{\mathsf{HOTCom}}$ and otherwise follow the protocol. If any $c_j$ was not the output of $\mathcal{F}_{\mathsf{GpRO}}$ on input $x_j$ or if any $x_j$ preimage is different from the $m_j$ $\mathcal{P}_{\mathrm{S}}$ should have sent based on the values extracted during **Commit** then set the flag $\texttt{cheated1} \leftarrow \top$.

**Delayed Hint** Upon receiving the broadcast $(\mathsf{cid}_j, c_j)_{j\in[k]}$ from $\mathcal{P}_{\mathrm{S}}$, find the preimage $x_j$ for each $c_j$ under the random oracle. If any $c_j$ does not have a preimage under $\mathcal{F}_{\mathsf{GpRO}}$ or there are multiple preimages then let $x_j$ be a random value in $\mathbb{F}_p$. If $x_j \neq m_j$ where $m_j$ is the value of the commitment $\mathsf{cid}_j$ based on what was extracted during **Commit** then set $\texttt{cheated2} \leftarrow \top$. Send $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j, x_j)_{j\in[k]})$ to $\mathcal{F}_{\mathsf{HOTCom}}$ and otherwise follow the protocol.

**Open** Upon receiving the broadcast $A_{\texttt{pub}}, A_{\texttt{delay}}, \mathbf{S}^*, \mathbf{D}^*$ as well as the private $B_{\texttt{priv}}$ do the following:

1. Run the same steps as $\mathcal{P}_{\mathrm{R}}$ runs in the protocol for **Open**. If any of the $c_j$ of **Hint** did not have a preimage before the call or is incorrect, then set $\texttt{cheated1} \leftarrow \top$. If $\texttt{cheated1} = \top$ or there is any $\mathcal{F}_{\mathsf{GpRO}}$ output queried by $\mathcal{A}$ that has more than two inputs (a collision) then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$.

2. Perform the checks in **Verify**. If any of them fails then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$. Otherwise send $(\mathsf{Open}, \mathsf{sid})$.

**Delayed Open** Upon receiving the broadcast $A_{\texttt{delay,open}}$ do the following:

1. Run the same steps as $\mathcal{P}_{\mathrm{R}}$ runs in the protocol for **Delayed Open**. If any of the $c_j$ for **Delayed Open** did not have a preimage before the call or is incorrect, then set $\texttt{cheated2} \leftarrow \top$. If $\texttt{cheated2} = \top$ or there is any $\mathcal{F}_{\mathsf{GpRO}}$ output queried by $\mathcal{A}$ that has more than two inputs (a collision) then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$.

2. Perform the checks in **Verify**. If any of them fails then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$. Otherwise send $(\mathsf{Open}, \mathsf{sid})$.

**Verify, Audit and Challenges** Send the command to $\mathcal{F}_{\mathsf{HOTCom}}$ and output what it outputs.

The simulation and the protocol can only be distinguished if $\mathcal{S}_{\mathsf{HOT-1-HonestReceiver}}$ aborts when $\mathcal{P}_{\mathrm{R}}$ would not in the real protocol. This happens when preimages of $\overline{\Delta}_1$ or some $c_j$ are undefined but nevertheless opened correctly later, or if the adversary found any collision on a random oracle output, or if any $x_j$ revealed during **Hint** or **Delayed Hint** was different from the previously extracted $m_j$. For the random oracle collisions or preimages that did not exist, we can bound the probability of this event by Proposition 1 as $Q^2/2^{2\lambda}$ if $Q$ random oracle calls are made by $\mathcal{A}$. Now, assume that the adversary was able to open $x_j \neq m_j$ and make the protocol **Verify** be correct, while $\mathcal{S}_{\mathsf{HOT-1-HonestReceiver}}$ aborts.

For the sake of argument, assume that $\mathbf{S}^* = \mathbf{S}$ that $\mathcal{P}_{\mathrm{S}}$ should have sent in the protocol. Since all the $g_j$ are fixed before $\Delta_1$ is uniformly chosen by the random oracle (and since $\mathbf{S}$ is correct), the chance that $\Delta_1$ is chosen such that the check passes is $1/p$ by Lemma 1. However, note that $\mathcal{A}$ may perform up to $Q$ queries to $\mathcal{F}_{\mathsf{GpRO}}$ with different choices of $g_j$ before outputting any response to $\mathcal{S}_{\mathsf{HOT-1-HonestReceiver}}$. Thus, the probability that it finds an accepting $\Delta_1$ during these calls is $Q/p$ by a union bound.

However, it could be that $\mathbf{S}^* \neq \mathbf{S}$ or $\mathbf{D}^* \neq \mathbf{D}$. Let $\mathbf{S}_E = \mathbf{S}^* - \mathbf{S}$ and $\mathbf{D}_E = \mathbf{D}^* - \mathbf{D}$. Since $\mathbf{D} = \mathbf{Q}_2 + \mathbf{Q}_1\Delta_1 - \mathcal{C}(\mathbf{S}) \cdot \mathsf{diag}(\Delta_0)$ and $\mathbf{D}^* = \mathbf{Q}_2 + \mathbf{Q}_1\Delta_1 - \mathcal{C}(\mathbf{S}^*) \cdot \mathsf{diag}(\Delta_0)$ we have that $\mathbf{D}^*_E = -\mathcal{C}(\mathbf{S}^*_E) \cdot \mathsf{diag}(\Delta_0)$. Since $\mathbf{S}^*_E \neq 0$ by assumption, $\mathcal{C}(\mathbf{S}^*_E)$ is non-zero in at least $d_{\mathcal{C}}$ positions, in which $\mathcal{A}$ would have had to predict the entries in $\Delta_0$ to fulfill the equation. $\Delta_0$ is chosen as an unpredictable output of **Delta Test** or **Init Done** (whose input $m$ uniquely fixes the transcript) to a query $\mathcal{A}$ might have made and that includes $\mathbf{S}^*, \mathbf{D}^*$, and for each query the chance of having guessed $\Delta_0$ in the right coordinates correctly is $1/N^{d_{\mathcal{C}}}$ for each attempt. Overall, the adversary can cheat in this check with probability $Q/N^{d_{\mathcal{C}}}$ by a union bound over all queries made to **Delta Test**.

Finally, since **Audit** outputs what **Verify** outputs, it will also only be incorrect when the real protocol succeeds but the simulation aborts. This has been analyzed above. **Challenges** is computed identically as how they are generated during **Commit** making them consistent between protocol and simulation.

$\mathcal{S}_{\mathsf{HOT-1-BothCorrupted}}$. The simulator can trivially be constructed from $\mathcal{S}_{\mathsf{HOT-1-HonestReceiver}}$. It again runs $\mathcal{F}_{\mathsf{WVOLE}}$ towards both the dishonest sender and receiver and the same techniques for input extraction and consistency checking still apply as communication only happens from $\mathcal{P}_{\mathrm{S}}$ to $\mathcal{P}_{\mathrm{R}}$ via $\mathcal{F}_{\mathsf{GBB}}$, privately or through $\mathcal{F}_{\mathsf{WVOLE}}$. The simulator can extract all openings except **Hint** with $i = 1$ but since the output of it is not verifiable by $\mathcal{P}_{\mathrm{A}}$ anyways, this does not pose a problem.

The security argument follows the same way as above, and we must only focus on **Audit**. However, the verification algorithm of **Audit** is identical to **Verify** (except that it skips private outputs to $\mathcal{P}_{\mathrm{R}}$). So the argument concerning the unpredictability of $\Delta_0$ still applies: $\mathcal{P}_{\mathrm{S}}$ and $\mathcal{P}_{\mathrm{R}}$ only learn $\Delta_0$ after all other messages defining $A_{\mathtt{pub}}, A_{\mathtt{priv}}, A_{\mathtt{delay}}, \Delta_1, \mathbf{S}, \mathbf{D}$ have been broadcast and $\mathbf{U}, \mathbf{V}$ have been fixed, which makes the transcript verifiable by $\mathcal{P}_{\mathrm{A}}$.

# F Multi-Receiver commitment protocol and proof

We describe the protocol $\Pi^{1-n}_{\mathsf{HOTCommit}}$ in Figures 20 and 21.

## F.1 Proof of Theorem 2

We prove Theorem 2 by providing two simulators. The simulator $\mathcal{S}_{\mathsf{HOTCCorruptSender}}$ shown in Figure 22 and Figure 23 assumes that the sender $\mathcal{P}_{\mathrm{S}}$ and a subset of the receivers $\mathcal{P}_{\mathrm{R}}$ are corrupt, while the simulator in $\mathcal{S}_{\mathsf{HOTCHonestSender}}$ shown in Figure 24 assumes the sender is honest and some or all $\mathcal{P} \in \mathcal{P}_{\mathrm{R}}$ are corrupt.

*Proof.*

*Corrupt sender.* When the sender $\mathcal{P}_{\mathrm{S}}$ is corrupt the adversary can cheat in two ways. Either by using hints that are different to the commitment values or by sending inconsistent values during **Commit** to the different instances of $\mathcal{F}^i_{\mathsf{HOTCom}}$.

During **Initialize** $\mathcal{S}$ will start local instances of $\mathcal{F}^i_{\mathsf{HOTCom}}$ to simulate communication with corrupt parties and send init to $\mathcal{F}_{\mathsf{HOTCom}}$ with which honest parties will communicate. In **Commit** the sender will send one message to each instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$ for every $i \in [|\mathcal{P}_{\mathrm{R}}|]$. If $\mathcal{P}_{\mathrm{S}}$ is cheating by sending inconsistent values $\mathcal{S}$ will set $\mathtt{fl}_{\mathrm{cheated}} = \top$. Although $\mathcal{S}$ detects cheating during **Commit**, it only aborts the execution based on that during **Open**, in order to maintain consistency with $\Pi^{1-n}_{\mathsf{HOTCommit}}$. The simulator will take the values sent to $\mathcal{F}^1_{\mathsf{HOTCom}}$ as reference for the remainder of the simulation (assuming that these are the extracted inputs) as we are only interested if $\mathcal{P}_{\mathrm{S}}$ cheated or not. When commit in each $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance is completed it will output a random value $\overline{\mathbf{ch}}^i$. The value is sampled uniformly at random for each commitment and thus unpredictable. In $\Pi^{1-n}_{\mathsf{HOTCommit}}$ these values are then combined to $\mathbf{ch}$ by querying $\mathcal{F}_{\mathsf{GpRO}}$ on message $\overline{\mathbf{ch}}^1 \| \ldots$. To be consistent with $\mathcal{F}_{\mathsf{HOTCom}}$ the simulator has to program $\mathcal{F}_{\mathsf{GpRO}}$ such that a query with $\overline{\mathbf{ch}}^1 \| \ldots$ will output the same as $\mathcal{F}_{\mathsf{HOTCom}}$. However, an adversary may have queried this input before the $\overline{\mathbf{ch}}^i$ are output by $\mathcal{F}^i_{\mathsf{HOTCom}}$ and then programming is not possible, making the simulator abort and the simulation distinguishable. The probability of this event is $\frac{Q^2}{2^{2\lambda}}$ as given by Proposition 1, because each $\overline{\mathbf{ch}}^i$ comes from $\{0,1\}^{2\lambda}$.

When **Hint** is executed $\mathcal{P}_{\mathrm{S}}$ will send the value to $\mathcal{F}_{\mathsf{GBB}}$ and all $\mathcal{F}^i_{\mathsf{HOTCom}}$ instances. $\mathcal{S}$ will compare the value that is send to $\mathcal{F}_{\mathsf{GBB}}$ with the value the $\mathcal{F}^i_{\mathsf{HOTCom}}$ instances output and abort if these do not match. The value that was sent via $\mathcal{F}_{\mathsf{GBB}}$ is given to $\mathcal{F}_{\mathsf{HOTCom}}$.

For **Delayed Hint** we extract the values from the commitment *com* that the adversary broadcasts. Delayed values are opened in **Delayed Open** when the commitment is opened and the opening verified. Here the adversary can cheat by finding another preimage to $com \in \{0,1\}^{2\lambda}$, which it can do in $Q$ queries to

$\Pi^{1-n}_{\mathsf{HOTCommit}}$ **(Part 1)**

In the protocol, a sender $\mathcal{P}_{\mathrm{S}}$ interacts with a set of receivers $\mathcal{P}_{\mathrm{R}}$ and auditor $\mathcal{P}_{\mathrm{A}}$. All parties have access to an instance of $\mathcal{F}_{\mathsf{GpRO}}$ and $\mathcal{F}_{\mathsf{GBB}}$. $\mathcal{F}_{\mathsf{HOTCom}}$ is used in the $|\mathcal{P}_{\mathrm{R}}| = 1$ setting. The $\mathcal{F}_{\mathsf{HOTCom}}$ instance between $\mathcal{P}_{\mathrm{S}}$ and a party $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$ is denoted $\mathcal{F}^i_{\mathsf{HOTCom}}$. Let $\mathcal{T} = \{0,1\}^{2\lambda}$ for $\mathcal{F}^i_{\mathsf{HOTCom}}$.

**Init** On input $(\mathsf{Init}, \mathsf{sid}, \mathrm{mc})$ to all parties, initialize one instance of $\mathcal{F}_{\mathsf{HOTCom}}$ for every party $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$.

1. Send $(\mathsf{Init}, \mathsf{sid}, \mathrm{mc}+1)$ to each instance of $\mathcal{F}_{\mathsf{HOTCom}}$.
2. $\mathcal{P}_{\mathrm{S}}$ samples $r \leftarrow \mathbb{F}_p$ at random.
3. Send $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{cid}_{\mathrm{mc}+1}, r))$ to each instance $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $i \in [|\mathcal{P}_{\mathrm{R}}|]$.

**Commit** On input $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{cid}_j, m_j)_{j\in[k]}))$, $\mathcal{P}_{\mathrm{S}}$ sends $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{cid}_j, m_j)_{j\in[k]}))$ to all instances of $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $i \in [|\mathcal{P}_{\mathrm{R}}|]$. It then obtains $\overline{\mathbf{ch}}^i$ that was returned by each $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance.

1. Each party $\mathcal{P}_j$ obtains all $\overline{\mathbf{ch}}^i$ that were returned by each $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance
2. Every party sends $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}}^1 \| \ldots \| \overline{\mathbf{ch}}^{|\mathcal{P}_{\mathrm{R}}|}, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and outputs the returned value $\mathbf{ch}$.

**Hint** On input $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, i)$ to $\mathcal{P}_{\mathrm{S}}$ where each $(\mathsf{cid}_j, m_j)$ was sent to **Commit** before: If $i \neq 0$ then $\mathcal{P}_{\mathrm{S}}$ sends $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, 1)$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$. If $i = 0$ then:

1. $\mathcal{P}_{\mathrm{S}}$ sends $(\mathsf{Broadcast}, (\mathsf{cid}_j, m_j)_{j\in[k]})$ to $\mathcal{F}_{\mathsf{GBB}}$.
2. Additionally, $\mathcal{P}_{\mathrm{S}}$ sends $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, 0)$ to all instances of $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $i \in [|\mathcal{P}_{\mathrm{R}}|]$.
3. Each receiver $\mathcal{P}_i$ compares $(\mathsf{cid}_j, m_j)_{j\in[k]}$ that was sent to $\mathcal{F}_{\mathsf{GBB}}$ with the value $(m'_j)_{j\in[k]}$ that is the output of **Hint**. If $m_j \neq m'_j$ then broadcast $(\mathsf{Abort}, \mathsf{sid})$ and abort the protocol.

**Delayed Hint** On input $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$, $\mathcal{P}_{\mathrm{S}}$ does the following:

1. Sample $r \leftarrow \{0,1\}^\lambda$ uniformly at random.
2. Send $(\mathsf{Query}, \mathsf{sid}, (\mathsf{cid}_j, m_j)_{j\in[k]} \| r, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain *com*.
3. Broadcast $(\mathsf{Broadcast}, com)$ and send $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$ to all $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $i \in [|\mathcal{P}_{\mathrm{R}}|]$.

**Open** On first input $(\mathsf{Open}, \mathsf{sid})$, $\mathcal{P}_{\mathrm{S}}$ does the following:

1. Run the procedure **Consistency Check**.
2. $\mathcal{P}_{\mathrm{S}}$ sends $(\mathsf{Open}, \mathsf{sid})$ to every instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $i \in [|\mathcal{P}_{\mathrm{R}}|]$.

Every $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$ sends $(\mathsf{Verify}, \mathsf{sid}, \mathsf{cid}_h, h)$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$. If the check passes then it continues, otherwise it broadcasts $(\mathsf{Broadcast}, \mathit{Abort})$ and aborts.

**Delayed Open** On second input $(\mathsf{Open}, \mathsf{sid})$, $\mathcal{P}_{\mathrm{S}}$ does the following:

1. Send $(\mathsf{Broadcast}, (\mathsf{cid}_j, m_j)_{j\in[k]}), r)$ for the $\mathsf{cid}_j$ sent in **Delayed Hint**.
2. $\mathcal{P}_{\mathrm{S}}$ sends $(\mathsf{Open}, \mathsf{sid})$ to every instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $i \in [|\mathcal{P}_{\mathrm{R}}|]$.

Thereupon, each party in $\mathcal{P}_{\mathrm{R}}$ sends $(\mathsf{Query}, \mathsf{sid}, (\mathsf{cid}_j, m_j)_{j\in[k]}) \| r, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and compares the result with *com*. If it is not identical, then broadcast $(\mathsf{Broadcast}, \mathit{abort})$ and abort.

Fig. 20: Protocol $\Pi^{1-n}_{\mathsf{HOTCommit}}$ implementing multi-receiver one-time commitments.

$\Pi^{1-n}_{\mathsf{HOTCommit}}$ **(Part 2)**

**Add** On input $(\mathsf{Add}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[3]})$, forward the message to every instance $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $i \in [|\mathcal{P}_{\mathrm{R}}|]$.

**Linear** On input $(\mathsf{Linear}, \mathsf{sid}, \mathsf{cid}_1, \mathsf{cid}_2, \alpha, \delta)$, forward the message to every instance $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $i \in [|\mathcal{P}_{\mathrm{R}}|]$.

**Verify** On input $(\mathsf{Verify}, \mathsf{sid}, L)$ to any $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$, send $(\mathsf{Verify}, \mathsf{sid}, L)$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$ and output what it outputs.

**Audit** On input $(\mathsf{Audit}, \mathsf{sid}, (\mathsf{cid}_j, f_j)_{j\in[k]})$ to $\mathcal{P}_{\mathrm{A}}$ where $f_j$ are $\mathbb{F}_p$-linear functions created by **Add** and **Linear**:

1. Send $(\mathsf{Challenges}, \mathsf{sid})$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$. Let $\overline{\mathbf{ch}}^i$ be the last challenge on each returned list. Then send $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}}^1 || \ldots || \overline{\mathbf{ch}}^{|\mathcal{P}_{\mathrm{R}}|}, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and obtain the response $\mathbf{ch}$.
2. Send $(\mathsf{Query}, \mathsf{sid}, \mathbf{ch}, \mathbb{F}_p^{\mathrm{mc}})$ to get $c_1, \ldots, c_{\mathrm{mc}}$.
3. Send $(\mathsf{Audit}, \mathsf{sid}, \mathsf{cid}_j, \sum_{j\in[k]} c_j \cdot \mathsf{cid}_j + \mathsf{cid}_{\mathrm{mc}+1})$ to every instance $\mathcal{F}^i_{\mathsf{HOTCom}}$. If they return different messages $(\mathsf{Audit}, \mathsf{sid}, L)$ then return $(\mathsf{AuditFailed}, \mathsf{sid})$.
4. Send $(\mathsf{Audit}, \mathsf{sid}, (\mathsf{cid}_j, f_j)_{j\in[k]})$ to every $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance for $i \in [|\mathcal{P}_{\mathrm{R}}|]$. If all output the same $(\mathsf{Audit}, \mathsf{sid}, L)$ then output $(\mathsf{Audit}, \mathsf{sid}, L)$, otherwise $(\mathsf{AuditFailed}, \mathsf{sid})$.

**Challenges** On input $(\mathsf{Challenges}, \mathsf{sid})$ to $\mathcal{P}_{\mathrm{A}}$:

1. For each $i \in \mathcal{P}_{\mathrm{R}}$ send $(\mathsf{Challenges}, \mathsf{sid})$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$ and obtain $L_i$.
2. Compute each $\mathbf{ch}_i$ as any receiver computes it in **Commit** and output all such values.

**Procedure Consistency Check** Subroutine to verify that commitments and their openings are consistent.

1. Let $\mathbf{ch}$ be the most recent challenge. Send $(\mathsf{Query}, \mathsf{sid}, \mathbf{ch}, \mathbb{F}_p^{\mathrm{mc}})$ to $\mathcal{F}_{\mathsf{GpRO}}$ and denote the output as $c_1, \ldots, c_{\mathrm{mc}}$.
2. $\mathcal{P}_{\mathrm{S}}$ computes $cc = r + \sum_{j=1}^{\mathrm{mc}} c_j \cdot m_j$ and sends $(\mathsf{Broadcast}, cc)$ to $\mathcal{F}_{\mathsf{GBB}}$.
3. Compute the function on the previous steps on every $\mathcal{F}^i_{\mathsf{HOTCom}}$, using $\mathsf{cid}_r$ and $\mathsf{cid}_r, \ldots, \mathsf{cid}_{\mathrm{mc}}$. Denote the resulting commitment ID as $\mathsf{cid}_{cc}$.
4. $\mathcal{P}_{\mathrm{S}}$ sends $(\mathsf{Hint}, \mathsf{sid}, \mathsf{cid}_{cc}, 0)$ to every instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$.
5. Each $\mathcal{P}_i \in \mathcal{P}_{\mathrm{R}}$ verifies that $m_{cc}$ received from $\mathcal{F}^i_{\mathsf{HOTCom}}$ equals $cc$. If $m_{cc} \neq cc$ send $(\mathsf{Broadcast}, \mathit{Abort})$ to $\mathcal{F}_{\mathsf{GBB}}$ and output 0.

Fig. 21: Protocol $\Pi^{1-n}_{\mathsf{HOTCommit}}$ implementing multi-receiver one-time commitments.

RO with probability at most $\frac{Q^2}{2^{2\lambda}}$. For **Add, Linear** the simulation is straight forward as the simulator follows the steps of the protocol.

**Open** the parties perform a consistency check that uses outputs from $\mathcal{F}_{\mathsf{GpRO}}$ and the universal hash function defined in Proposition 2. Note that the challenges $c_1, \ldots, c_{\mathrm{mc}}$ used in the **Consistency Check** are chosen based on all $\overline{\mathbf{ch}}^i$, which are only generated once the commitments (which may be different for each $\mathcal{F}^i_{\mathsf{HOTCom}}$) have been sent. The probability that $\mathcal{A}$ has made the query to $\mathcal{F}_{\mathsf{GpRO}}$

that generates the challenges $c_1, \ldots, c_{\text{mc}}$ before the $\overline{\mathbf{ch}}^i$ are chosen is bounded by $\frac{Q^2}{2^{2\lambda}}$ as $\overline{\mathbf{ch}}^i \in \{0,1\}^{2\lambda}$. During the consistency check $\mathcal{P}_\text{S}$ will open the value $m_{cc}$ which, if commitments are consistent throughout the $\mathcal{F}^i_{\mathsf{HOTCom}}$ instances, is equal to the value $h$ that was broadcast. The simulator will always abort if the commitments were inconsistent, while the protocol continues if $\mathcal{A}$ found a collision for the universal hash function. The universal hash function is fixed by the random challenge, and the probability of a collision is $\frac{1}{p}$ for each query. This gives a $\frac{Q}{p}$ distinguishing advantage for $\mathcal{Z}$ on this event, or $\frac{Q}{p} + \frac{Q^2}{2^{2\lambda}}$ in total.

*Sender and all receivers corrupt.* When all parties including the sender are dishonest the simulator will follow the steps of the *dishonest sender simulator* to extract the inputs of the sender. In the simulation, the simulator will always abort (and not extract inputs) if $\mathcal{A}$ attempts to cheat. In the protocol, the parties will output $(\mathsf{VerifyFailed}, \mathsf{sid})$ if $\mathcal{A}$ cheats by sending inconsistent values and the consistency check fails, which reduces to the same distinguishing advantage as mentioned above.

*Honest sender.* When the sender is honest and a subset or all $\mathcal{P}_i \in \mathcal{P}_\text{R}$ are corrupt $\mathcal{S}$ will simulate the outputs of every $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance. During **Init** $\mathcal{S}$ will follow the steps of $\Pi^{1-n}_{\mathsf{HOTCommit}}$ such that there exists an instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$ for every $\mathcal{P}_i \in \mathtt{C}$. When $\mathcal{S}$ receives $(\mathsf{Receipt}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, \mathbf{ch})$ they send a commitment message to each $\mathcal{F}^i_{\mathsf{HOTCom}}$ using 0 as a dummy value. This is indistinguishable for any $\mathcal{P}_i \in \mathtt{C}$ as their instance will only output the commitment ID. To replicate the challenge value $\mathbf{ch}$ (which will be the output of $\mathcal{F}_{\mathsf{HOTCom}}$) the simulator chooses random $\overline{\mathbf{ch}}^i$ values to $\mathcal{F}_{\mathsf{GBB}}$ and programs $\mathcal{F}_{\mathsf{GpRO}}$ such that it outputs $\mathbf{ch}$ when queried on $\overline{\mathbf{ch}}^1 \| \ldots$. Since each $\overline{\mathbf{ch}}^i$ is in $\{0,1\}^{2\lambda}$ and the adversary makes at most $Q$ queries to $\mathcal{F}_{\mathsf{GpRO}}$, the programming can fail with probability at most $Q^2/2^{2\lambda}$. For **Add, Linear, Verify** and **Audit** the simulation is straight forward as they are non-interactive. $\mathcal{S}$ simply lets every $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance output what was send by $\mathcal{F}_{\mathsf{HOTCom}}$. For **Hint** $\mathcal{S}$ has to broadcast the value received from $\mathcal{F}_{\mathsf{HOTCom}}$ first. For **Delayed Hint** it broadcasts a random commitment. When **Open** is executed $\mathcal{S}$ will first perform the consistency check and open $cc$ to the value send to $\mathcal{F}_{\mathsf{GBB}}$. When the sender is honest the functionality and protocol are not distinguishable by any environment $\mathcal{Z}$. The value $cc$ is indistinguishable from the protocol because the chosen universal hash function is hiding as per Definition 2. During **Delayed Open** the simulator equivocates the simulated commitment to the real openings $L$ by programming the random oracle on a random input. During $Q$ queries to the random oracle before programming (thus leading to an abort), the adversary could have guessed $r$ with probability at most $Q/2^\lambda$.

**$\mathcal{S}_{\mathsf{HOTCCorruptSender}}$ (Part 1)**

$\mathtt{H}$ denotes the honest parties and $\mathtt{C}$ the corrupt parties. Every $\mathcal{P}_i$ is connected to one instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$. $\mathcal{S}$ will simulate all $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $\mathcal{P}_i \in \mathtt{H}$. Let $m$ denote the number of honest parties. Note that **Verify**, **Challenges** and **Audit** are non-interactive protocols and need no simulation.

On input $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}}^1 \| \ldots, \mathcal{T})$ to $\mathcal{F}_{\mathsf{GpRO}}$ check if each $\overline{\mathbf{ch}}^i$ was output by $\mathcal{F}^i_{\mathsf{HOTCom}}$. If so, if the same $(\mathsf{cid}_j, m_j)_{j\in[k]}$ were sent to all $\mathcal{F}^i_{\mathsf{HOTCom}}$ instances and if all $\overline{\mathbf{ch}}^i$ were output to the same input by all $\mathcal{F}^i_{\mathsf{HOTCom}}$ then send all $(\mathsf{cid}_j, m_j)_{j\in[k]}$ to **Challenge Test** of $\mathcal{F}_{\mathsf{HOTCom}}$ and let $\mathcal{F}_{\mathsf{GpRO}}$ output what it outputs. Otherwise simulate $\mathcal{F}_{\mathsf{GpRO}}$ honestly.

**Init**
1. Set $\mathtt{fl}_{\text{cheated}} = \bot$.
2. Start local instance of $\Pi^{1-n}_{\mathsf{HOTCommit}}$, such that there exists one instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$ for every party.
3. Run all steps of **Init** in $\Pi^{1-n}_{\mathsf{HOTCommit}}$ and simulate honest parties $\mathcal{P} \in \mathtt{H}$.
4. Send $(\mathsf{Init}, \mathsf{sid}, \text{mc})$ to $\mathcal{F}_{\mathsf{HOTCom}}$ if **Init** in $\Pi^{1-n}_{\mathsf{HOTCommit}}$ finishes.

**Commit** $\mathcal{P}_{\text{S}}$ will send one message for each instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$. Let $M^i$ be the message $(\mathsf{Commit}, \mathsf{sid}, n, (\mathsf{cid}_j, m_j)_{j\in[k]})$ intended for $\mathcal{P}_i \in \mathcal{P}_{\text{R}}$.
1. Wait until $\mathcal{P}_{\text{S}}$ sends all commit messages $M^1, \ldots, M^{|\mathcal{P}_{\text{R}}|}$, where $M^1$ is intended for $\mathcal{F}^i_{\mathsf{HOTCom}}$.
2. For every message $M_{i>1}$ that was sent by $\mathcal{P}_{\text{S}}$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$, compute $\delta = M^1 - M^i$. If $\delta \neq 0$ for any of the messages, set $\mathtt{fl}_{\text{cheated}} = \top$.
3. Send $M^1$ to $\mathcal{F}_{\mathsf{HOTCom}}$ and wait for the challenge output **ch**. Program $\mathcal{F}_{\mathsf{GpRO}}$ such that it will output **h** when queried on $\overline{\mathbf{ch}}^1 || \ldots || \overline{\mathbf{ch}}^{|\mathcal{P}_{\text{R}}|}$. If the programming fails, then abort.

**Hint** On input $(\mathsf{Hint}, \mathsf{sid}, \mathsf{cid}_j)_{j\in[k]}, 0)$ by $\mathcal{P}_{\text{S}}$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$:
1. Wait for $\mathcal{P}_{\text{S}}$ to broadcast $(\mathsf{cid}_j, x_j)_{j\in[k]}$.
2. Wait for input $(\mathsf{Hint}, \mathsf{sid}, (m_j)_{j\in[k]})$ sent by $\mathcal{A}$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$ with $\mathcal{P}_i \in \mathtt{H}$.
3. If any $m_j \neq x_j$ send $(\mathsf{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$ and $(\mathsf{Broadcast}, \mathit{abort})$ to $\mathcal{F}_{\mathsf{GBB}}$.
4. Send $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, 0)$ and $(\mathsf{Hint}, \mathsf{sid}, (x_j)_{j\in[k]})$ to $\mathcal{F}_{\mathsf{HOTCom}}$.

On input $(\mathsf{Hint}, \mathsf{sid}, \mathsf{cid}_j)_{j\in[k]}, 1)$ by $\mathcal{P}_{\text{S}}$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$ where $i \in \mathtt{H}$:
1. Wait for input $(\mathsf{Hint}, \mathsf{sid}, (m_j)_{j\in[k]})$ sent by $\mathcal{A}$ to $\mathcal{F}^i_{\mathsf{HOTCom}}$.
2. Send $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, i)$ to $\mathcal{F}_{\mathsf{HOTCom}}$
3. Input $(m^j)_{j\in[k]}$ to $\mathcal{F}_{\mathsf{HOTCom}}$.

Fig. 22: Simulator of $\mathcal{F}_{\mathsf{HOTCom}}$ with parameter $|\mathcal{P}_{\text{R}}| > 1$ and corrupt sender.

$\mathcal{S}_{\mathsf{HOTCCorruptSender}}$ **(Part 2)**

**Delayed Hint** On input $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$ by $\mathcal{P}_\mathrm{S}$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$:

1. Wait for the broadcast $(\mathsf{Broadcast}, com)$ on $\mathcal{F}_{\mathsf{GBB}}$. Let $(\mathsf{cid}_j, m_j)_{j\in[k]} \| r$ be the preimage under $\mathcal{F}_{\mathsf{GpRO}}$. If any value is undefined, then let $m_j$ be uniformly random.
2. Wait for input $(\mathsf{Hint}, \mathsf{sid}, (m_j)_{j\in[k]})$ sent by $\mathcal{A}$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$ with $\mathcal{P}_i \in \mathtt{H}$.
3. Send $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$ and $(\mathsf{Hint}, \mathsf{sid}, (m_j)_{j\in[k]})$ to $\mathcal{F}_{\mathsf{HOTCom}}$.

**Open** On first input $(\mathsf{Open}, \mathsf{sid})$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$ by $\mathcal{P}_\mathrm{S}$:

1. Follow the **Consistency Check** procedure in $\Pi^{1-n}_{\mathsf{HOTCommit}}$. If $m_h \neq h$ abort.

**Delayed Open** On second input $(\mathsf{Open}, \mathsf{sid})$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$ by $\mathcal{P}_\mathrm{S}$: Wait for $\mathcal{P}_\mathrm{S}$ to broadcast $L', r'$ and recompute $com$. If any message in $L'$ does not coincide with the messages committed in $com$ or $r' \neq r$ then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$, otherwise send $(\mathsf{Open}, \mathsf{sid})$.

**Add** Follow the steps described in $\Pi^{1-n}_{\mathsf{HOTCommit}}$.
**Linear** Follow the steps described in $\Pi^{1-n}_{\mathsf{HOTCommit}}$.
**Verify** Follow the steps described in $\Pi^{1-n}_{\mathsf{HOTCommit}}$, if $\mathtt{fl}_{\mathrm{cheated}} = \top$ output $(\mathsf{VerifyFailed}, \mathsf{sid})$.

Fig. 23: Simulator of $\mathcal{F}_{\mathsf{HOTCom}}$ with parameter $|\mathcal{P}_\mathrm{R}| > 1$ and corrupt sender.

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

In the following the sender $\mathcal{P}_{\mathrm{S}}$ is honest and the set of all receivers is denoted as $\mathcal{P}_{\mathrm{R}}$. $\mathtt{H}$ denotes honest receivers and $\mathtt{C}$ the corrupt receivers. If any dishonest party broadcasts $(\mathsf{Broadcast}, \mathit{abort})$, then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{HOTCom}}$. Note that **Verify**, **Challenges** and **Audit** are non-interactive and require no simulation.

**Init** Upon $(\mathsf{Init}, \mathsf{sid}, \mathrm{mc})$ from $\mathcal{F}_{\mathsf{HOTCom}}$:
1. Start local instance of $\Pi^{1-n}_{\mathsf{HOTCommit}}$ with which the corrupt parties will communicate. Therefore, each $\mathcal{P} \in \mathtt{C}$ will have access to its own instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$ where the $\mathcal{P}_{\mathrm{S}}$ is simulated by $\mathcal{S}$. The instance has $\mathrm{mc}+1$ commitments.
2. Simulate honest uncorrupted receivers towards the adversary.

**Commit** Upon message $(\mathsf{Receipt}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]}, \mathbf{ch})$ from $\mathcal{F}_{\mathsf{HOTCom}}$ $\mathcal{S}$ sends $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{cid}_j, 0)_{j\in[k]})$ to each $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance. It picks uniformly random $\overline{\mathbf{ch}}^i$ for each $\mathcal{F}^i_{\mathsf{HOTCom}}$ to output, and programs $\mathcal{F}_{\mathsf{GpRO}}$ to output $\mathbf{ch}$ on input $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}}^1 \| \ldots, \mathcal{T})$. If the programming fails, then abort.

**Hint** On input $(\mathsf{Hint}, \mathsf{sid}, (\mathsf{cid}_j, x_j)_{j\in[k]}, i)$ from $\mathcal{F}_{\mathsf{HOTCom}}$, if $i = 0$:
1. Send $(\mathsf{Broadcast}, x_j)_{j\in[k]})$ to $\mathcal{F}_{\mathsf{GBB}}$.
2. Make each $\mathcal{F}^i_{\mathsf{HOTCom}}$ output the value $x_j$ for ID $\mathsf{cid}_j$.

If $i \neq 0$, then let the instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$ where $\mathcal{P}_i$ is the receiver output the message received from $\mathcal{F}_{\mathsf{HOTCom}}$. Also respond to verification queries to $\mathcal{F}^i_{\mathsf{HOTCom}}$ with the obtained value.

**Delayed Hint** Upon message $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$ from $\mathcal{F}_{\mathsf{HOTCom}}$:
1. Let $\mathit{com} \in \{0,1\}^{2\lambda}$ be uniformly random. Broadcast $(\mathsf{Broadcast}, \mathit{com})$ on $\mathcal{F}_{\mathsf{GBB}}$.
2. Let each $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance output $(\mathsf{DelHint}, \mathsf{sid}, (\mathsf{cid}_j)_{j\in[k]})$.

**Open** Upon first message $(\mathsf{Open}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{HOTCom}}$, run the protocol honestly.

**Delayed Open** Upon second message $(\mathsf{Open}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{HOTCom}}$:
1. Obtain $(\mathsf{Open}, \mathsf{sid}, L)$ from $\mathcal{F}_{\mathsf{HOTCom}}$.
2. Sample a uniformly random $r \in \{0,1\}^\lambda$ and program $\mathcal{F}_{\mathsf{GpRO}}$ to output $\mathit{com}$ on input $L\|r$. If the programming fails, then abort.
3. Broadcast $(\mathsf{Broadcast}, L\|r)$ on $\mathcal{F}_{\mathsf{GBB}}$ and respond to all verification queries on $\mathcal{F}^i_{\mathsf{HOTCom}}$ for $L$ accordingly.

**Add** Wait for $\mathcal{F}_{\mathsf{HOTCom}}$ to send $(\mathsf{Add}, \mathsf{sid}, (\mathsf{cid}_i)_{[3]})$ and send $(\mathsf{Add}, \mathsf{sid}, (\mathsf{cid}_j)_{[3]})$ to each instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$.

**Linear** Wait for $\mathcal{F}_{\mathsf{HOTCom}}$ to send $(\mathsf{Linear}, \mathsf{sid}, \mathit{cid}_1, \mathit{cid}_2, \alpha, \delta)$ and send $(\mathsf{Linear}, \mathsf{sid}, \mathsf{cid}_1, \mathsf{cid}_2, \alpha, \delta)$ to each instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$.

**Verify** When **Verify** was send by $\mathcal{P}_i \in \mathtt{C}$ though an instance of $\mathcal{F}^i_{\mathsf{HOTCom}}$, forward the message to $\mathcal{F}_{\mathsf{HOTCom}}$ and let all $\mathcal{F}^i_{\mathsf{HOTCom}}$ instances output what $\mathcal{F}_{\mathsf{HOTCom}}$ outputs.

Fig. 24: Simulator of $\mathcal{F}_{\mathsf{HOTCom}}$ with parameter $|\mathcal{P}_{\mathrm{R}}| > 1$ and honest sender.

## G Proof of Theorem 3 (Offline Protocol)

In the proof, we mainly focus on the case with the presence of at least one honest party in $\mathcal{P}$, whose simulator $\mathcal{S}_{\mathsf{OffHonP}}$ we describe in Figure 25 and Figure 26. We then argue how the setting where every party in $\mathcal{P}$ is corrupted follows.

*Proof.* In the proof, we will focus on the case where there is one random value/triple and therefore drop the index $l$ to simplify notation.

*At least one honest party.* During **Initialize** the simulator initializes an instance of $\Pi_{\mathsf{Offline}}$. The simulator uses dummy values for all $\mathcal{P} \in \mathtt{H}$ throughout the protocol. For **Linear, Verify** the simulation is straightforward as the parties follow the steps in $\Pi_{\mathsf{Offline}}$.

For every party $\mathcal{P}_c \in \mathtt{C}$ the simulator stores the share $r_c$ that the party inputs into $\mathcal{F}^c_{\mathsf{HOTCom}}$ during **Input** and forwards them to $\mathcal{F}_{\mathsf{Offline}}$.

During the **Triples** protocol, the adversary $\mathcal{A}$ can add two types of errors to the honest sharings. The first is a multiplicative error, denoted by $\delta^h_a$ and $\delta^h_b$, which can differ for every honest party $\mathcal{P}_h$. The second is an additive term $\Delta_c$ for all dishonest parties.

During $\Pi_{\mathsf{Offline}}$ $\delta^h_a, \delta^h_b$ are introduced when one or more of the dishonest parties $\mathcal{P}_c$ send inputs to $\mathcal{F}_{\mathsf{OLE}}$ which differ from their additive shares $a^c, b^c$ that they have committed to in $\mathcal{F}_{\mathsf{HOTCom}}$ before. In the simulation this is modeled by going through each interaction of an honest party $\mathcal{P}_h$ with each party $\mathcal{P}_c \in \mathtt{C}$ and recording the difference between the values $\overline{a}^c, \overline{b}^c$ that the adversary inputs for $\mathcal{F}_{\mathsf{OLE}}$ and the values $a^c, b^c$ that were committed previously. Clearly, the difference $\overline{a}^c - a^c$ is multiplied with the honest $b^h$ when using $\mathcal{F}_{\mathsf{OLE}}$ and both parties obtain additive shares of the result, therefore it impacts the product as the aforementioned error. We therefore add this term to $\delta^h_b$, as well as any error introduced by any other dishonest party when multiplying with $b^h$. The same reasoning applies when computing $\delta^h_a$. Here, note that from the perspective of the adversary, while the shares of the honest party get multiplied with a value chosen by the attacker, making the error multiplicative makes it independent of the concrete choice of shares of the honest parties i.e. identical in simulation and the real protocol.

Furthermore, the **Triples** protocol will output a challenge value $h$. For every input to $\mathcal{F}^i_{\mathsf{HOTCom}}$ the adversary will see a uniformly random challenge output. These are then combined, in the protocol via $\mathcal{F}_{\mathsf{GpRO}}$, to a challenge to be output by the protocol. $\mathcal{S}$ will wait until $\mathcal{F}_{\mathsf{HOTCom}}$ sends $\mathbf{ch}$ as a response to all commitments and will program $\mathcal{F}_{\mathsf{GpRO}}$ to output $\mathbf{ch}$ when queried on $(\overline{\mathbf{ch}}^1 || \ldots || \overline{\mathbf{ch}}^{|\mathcal{P}|})$. Note that this can fail if $\mathcal{F}_{\mathsf{GpRO}}$ was already programmed to a different output, but this can only happen if $\mathcal{A}$ guessed $\overline{\mathbf{ch}}^i \in \{0,1\}^{2\lambda}$ before it was output by $\mathcal{F}^i_{\mathsf{HOTCom}}$. As the adversary makes at most $Q$ queries to $\mathcal{F}_{\mathsf{GpRO}}$, the distinguishing advantage due to this event is at most $Q^2/2^{2\lambda}$ by Proposition 1.

**Simulator $\mathcal{S}_{\mathsf{OffHonP}}$ (Part 1)**

In the following honest parties are denoted as $\mathtt{H}$ and corrupt parties as $\mathtt{C}$. The simulator $\mathcal{S}_{\mathsf{OffHonP}}$ maintains a copy of shares from all parties and updates them accordingly.

Upon query $(\mathsf{Query}, \mathsf{sid}, \overline{\mathbf{ch}}^1 \| \ldots, \mathbb{F}_p^*)$ to $\mathcal{F}_{\mathsf{GpRO}}$ and if all $\overline{\mathbf{ch}}^i$ were generated by $\mathcal{F}^i_{\mathsf{HOTCom}}$ as a last response to a commitment of shares for **Triples**, send all shares as input to **Triples** in $\mathcal{F}_{\mathsf{Offline}}$ and afterwards send $\mathsf{reject}$. Let $\mathcal{F}_{\mathsf{GpRO}}$ output what $\mathcal{F}_{\mathsf{Offline}}$ outputs. Otherwise, simulate $\mathcal{F}_{\mathsf{GpRO}}$ honestly.

**Initialize**

1. Start a local instance of $\Pi_{\mathsf{Offline}}$ with which the corrupt parties communicate. Simulate parties $\mathcal{P} \in \mathtt{H}$ and functionalities $\mathcal{F}_{\mathsf{HOTCom}}$ and $\mathcal{F}_{\mathsf{OLE}}$.
2. Run **Initialize** for all honest parties.
3. Upon obtaining $(\mathsf{Init}, \mathsf{sid}, \mathsf{in} + 3\mathsf{m})$ from $\mathcal{F}^j_{\mathsf{HOTCom}}$, $j \in \mathtt{C}$ send $(\mathsf{Init}, \mathsf{sid}, \mathsf{in}, \mathsf{m})$ to $\mathcal{F}_{\mathsf{Offline}}$.

**Input** Run the protocol honestly for each party in $\mathtt{H}$ and commit to random shares. Send shares that $\mathcal{P}_i \in \mathtt{C}$ inputs into $\mathcal{F}^i_{\mathsf{HOTCom}}$ to $\mathcal{F}_{\mathsf{Offline}}$.

**Triples**

1. For each $l \in [\mathsf{m}], \mathcal{P}_h \in \mathtt{H}$ set $\delta^h_{a,l} = \delta^h_{b,l} = 0$.
2. Follow step 1 of the protocol honestly. For the honest $a^h_l, b^h_k$ use the values generated by the simulator before. Let $a^c_l, b^c_l$ to be the values inserted into $\mathcal{F}^c_{\mathsf{HOTCom}}$ by the dishonest parties $\mathcal{P}_c \in \mathtt{C}$.
3. Simulate the interactions between each honest $\mathcal{P}_h$ and each corrupted $\mathcal{P}_c$ for each $l \in [\mathsf{m}]$ as follows:
   (a) Simulate calls to $\mathcal{F}_{\mathsf{OLE}}$ of $\mathcal{P}_h$ with values $a^h_l$ and $b^h_l$ defined above.
   (b) For each dishonest party $\mathcal{P}_c$, wait until $\mathcal{P}_c$ sends its multiplicative shares $(\overline{a}^c_{h,l}, \overline{b}^c_{h,l})$ to the instances of $\mathcal{F}_{\mathsf{OLE}}$.
   (c) Add $\overline{a}^c_{h,l} - a^c_l$ to $\delta^h_{a,l}$ and $\overline{b}^c_{h,l} - b^c_l$ to $\delta^h_{b,l}$.
   (d) Wait for each $\mathcal{P}_c$ to send $\overline{c}^c_l$ to $\mathcal{F}^c_{\mathsf{HOTCom}}$.
   (e) After the final commit wait for each $\mathcal{F}^i_{\mathsf{HOTCom}}$ instance to
4. Compute $c_l = (\sum_h a^h_l + \sum_c \overline{a}^c_l) \cdot (\sum_h b^h_l + \sum_c \overline{b}^c_l)$ and for every honest party set $c^h_l = a^h_l b^h_l + \sum_{j=1}^n (w^h_{j,l} + v^h_{j,l})$.
5. Set $\Delta_{c,l} = \sum_{c \in \mathtt{C}} \overline{c}^c_l - (c_l - \sum_{h \in \mathtt{H}} c^h_l)$.
6. When every interaction between all pairs of parties is completed as the last dishonest party commits to its share, input $\delta^h_{a,l}, \delta^h_{b,l}$ for each honest party, $a^c_l, b^c_l, \overline{c}^c_l$ for each dishonest party $\mathcal{P}_c$ as well as the cumulative $\Delta_{c,l}$. Then obtain $h$ as a response.
7. Before outputting $\overline{\mathbf{ch}}^i$ program $\mathcal{F}_{\mathsf{GpRO}}$ so that on input $(\overline{\mathbf{ch}}^1 || \ldots || \overline{\mathbf{ch}}^{|\mathcal{P}|})$ it will output $\mathbf{ch}$ (if its not already programmed to this value). If $\mathcal{F}_{\mathsf{GpRO}}$ cannot be reprogrammed, abort.

**Linear** For any $(\mathsf{Linear}, \mathsf{sid})$ sent by a dishonest party to $\mathcal{F}^j_{\mathsf{HOTCom}}$ send the respective command to $\mathcal{F}_{\mathsf{Offline}}$.

Fig. 25: Simulator for $\Pi_{\mathsf{Offline}}$ where at least one party is honest.

After all $\mathcal{F}_{\mathsf{OLE}}$ interactions are completed, all parties commit to their additive share of $c$ using $\mathcal{F}_{\mathsf{HOTCom}}$. Corrupt parties $\mathcal{P}_c$ can introduce an additive error $\Delta_c$

**Simulator $\mathcal{S}_{\mathsf{OffHonP}}$ (Part 2)**

**Open** All openings are simulated as follows:

1. Wait for the message $(\mathsf{Open}, \mathsf{sid}, \mathsf{id}, i, v)$ from $\mathcal{F}_{\mathsf{Offline}}$. Change the committed share $v_h$ of the simulated honest party $\mathcal{P}_h$ with the smallest index in $\mathcal{F}^h_{\mathsf{HOTCom}}$ such that $v = \sum_j v_j$.
2. Let all simulated honest parties run the protocol honestly & set $\delta = 0$.
3. Wait for each $\mathcal{P}_c \in \mathsf{C}$ to let $\mathcal{F}^j_{\mathsf{HOTCom}}$ release $(\mathsf{cid}^c, \overline{v}^c)$ (either towards an honest party or all parties).
4. Set $\Delta_c = \sum_{c \in C}(\overline{v}^c - v^c)$. If any $\overline{v}^c \neq v^c$ then set $\delta = 1$
5. Input $(\Delta_c, \delta)$ to $\mathcal{F}_{\mathsf{Offline}}$.
6. If instead $\mathsf{type} = 2$ then wait for all dishonest parties to send $(\mathsf{DelHint}, \mathsf{sid}, \mathsf{id})$ as well as $(\mathsf{Hint}, \mathsf{sid}, v^c)$ to each $\mathcal{F}^c_{\mathsf{HOTCom}}$, compute $\Delta_c, \delta$ as before and then send $(\mathsf{Open}, \mathsf{sid}, \mathsf{id}, 2, 0)$ and $\Delta_c, \delta$ to $\mathcal{F}_{\mathsf{Offline}}$.

**Release** Send $(\mathsf{Release}, \mathsf{sid})$ for each dishonest $\mathcal{P}_c$ upon first message $(\mathsf{Open}, \mathsf{sid})$ from $\mathcal{F}^c_{\mathsf{HOTCom}}$.

**Full Release**

1. Send $(\mathsf{Release}, \mathsf{sid})$ for each dishonest $\mathcal{P}_c$ upon first message $(\mathsf{Open}, \mathsf{sid})$ from $\mathcal{F}^c_{\mathsf{HOTCom}}$.
2. Upon obtaining $(\mathsf{Release}, \mathsf{sid}, \mathsf{id}, v)$ for each delayed $\mathsf{id}$ adjust the share $v_h$ of the simulated honest party $\mathcal{P}_h$ with the smallest index at $\mathcal{F}^h_{\mathsf{HOTCom}}$ such that $v = \sum_j v_j$.
3. Simulate the protocol honestly.

**Verify** Follow the steps of **Verify** in $\Pi_{\mathsf{Offline}}$.

**Challenge** Follow the steps of **Challenge** in $\Pi_{\mathsf{Offline}}$.

Fig. 26: Simulator for $\Pi_{\mathsf{Offline}}$ where at least one party is honest.

to the product by committing to a share $\overline{c}^c \neq c^c$ where $c^c$ is the value they should have committed to considering the outputs of $\mathcal{F}_{\mathsf{OLE}}$. To compute this additional additive error, note that $\mathcal{S}_{\mathsf{OffHonP}}$ has all shares of $a, b$ of both the simulated honest and the dishonest parties. We use the honest shares of $\mathcal{P}_h$ and the shares $\overline{a}^c, \overline{b}^c$ that the adversary uses for $\mathcal{F}_{\mathsf{OLE}}$ to define the product $c$ as it would be computed in the protocol at this point, including the errors $\delta^h_a, \delta^h_b$ introduced by the adversary. Thus, the output of the multiplication protocol using the inputs to $\mathcal{F}_{\mathsf{OLE}}$ can be written as

$$c = (\sum_h a^h + \sum_c \overline{a}^c) \cdot (\sum_h b^h + \sum_c \overline{b}^c)$$

for which the adversary can provide additive shares if it followed the protocol. However, the sum $\sum_h c^h + \sum_c \overline{c}^c$ may differ from $c$, and this difference defines the additive error term $\Delta_c$. Note that this computation of $\Delta_c$ removes all dependence on the $a^h, b^h, c^h$ of simulated honest parties and is therefore independent of the choice of the simulator. Hence, adding this constant error to the actual product in $\mathcal{F}_{\mathsf{Offline}}$ is not noticed by the adversary, and protocol and simulation are indistinguishable when the outcomes are reconstructed later.

During **Open** the simulator obtains from $\mathcal{F}_{\mathsf{Offline}}$ the correct openings, and adjusts the output of a single instance of $\mathcal{F}_{\mathsf{HOTCom}}$ towards $\mathcal{A}$ accordingly. It also waits for the openings from all $\mathcal{P}_c \in \mathtt{C}$. The sent values are compared to the values stored earlier, and the difference to the correct value is set as $\Delta_c$ that the simulator will input to $\mathcal{F}_{\mathsf{Offline}}$. In addition, it can happen that $\Delta_c = 0$ but shares of dishonest parties are still not opened correctly. In this case, $\mathcal{S}_{\mathsf{OffHonP}}$ will set $\delta = 1$ and send this to $\mathcal{F}_{\mathsf{Offline}}$ as well. During **Full Release**, the opening is equally equivocated to the correct value $v$ as during **Open**.

The openings that $\mathcal{S}_{\mathsf{OffHonP}}$ makes towards $\mathcal{A}$ have shares of the same distribution as in the protocol. Towards this, note that in both cases they reconstruct to the correct secret. If only one party is honest, then by definition the revealed share has the correct distribution. If more than one party is honest, for all but one of the parties the shares are distributed exactly as they are in the protocol. This is because we simulate the shares of each honest party (except the one we use for equivocation) by exactly following the protocol. For the last party, the distribution must be identical by correctness of the reconstruction. Moreover, **Verify** will output the same both during simulation and in the real protocol: in the real protocol, it outputs if dishonest parties open a share incorrectly. In the simulation, it does the same as $\mathcal{F}^h_{\mathsf{HOTCom}}$ for the equivocated honest party will always output that the equivocated share was valid to begin with.

*All parties corrupted.* In this case, the simulator only has to extract the shares for **Input** and **Triples** for input into $\mathcal{F}_{\mathsf{Offline}}$ as it simulates all instances of $\mathcal{F}_{\mathsf{HOTCom}}$ and otherwise just forwards messages to $\mathcal{F}_{\mathsf{Offline}}$ for completion of protocol steps. It also has to simulate $\mathcal{F}_{\mathsf{GpRO}}$ and performs this as in $\mathcal{S}_{\mathsf{OffHonP}}$. The shares sent to $\mathcal{F}^i_{\mathsf{HOTCom}}$ fully define which openings are correct and are input into $\mathcal{F}_{\mathsf{Offline}}$, and eventually the challenge $h$. In comparison to $\mathcal{S}_{\mathsf{OffHonP}}$ it can happen that a random oracle output for **Triples** may happen for **Input** shares that were not sent to $\mathcal{F}_{\mathsf{Offline}}$ as $\mathcal{A}$ "simulates" a protocol execution and the generated Fiat Shamir responses locally. To emulate this and the expected output challenges, $\mathcal{F}_{\mathsf{Offline}}$ allows the simulator to tentatively submit shares and then $\mathsf{reset}$ $\mathcal{F}_{\mathsf{Offline}}$ to try different shares afterwards. We use this to simulate responses correctly, and then let the simulator choose the correct response according the commitments of shares to all $\mathcal{F}^i_{\mathsf{HOTCom}}$ by $\mathcal{A}$. Note that while this seems to be a stronger adversarial capability, it still only leads to at most $Q$ random oracle queries that lead to reprogramming, and thus a distinguishing advantage of at most $Q^2/2^{2\lambda}$.

$\Pi_{\mathsf{Offline}}$ shows error during **Audit** exactly when a dishonest party either didn't open its share correctly or if the reconstructed sum is not equal to the input to the audit procedure. $\mathcal{F}_{\mathsf{Offline}}$ computes its result the same way, as incorrect openings in the simulation are forwarded by the simulator as $\Delta, \delta$ inputs during **Open** leading to the same abort during **Audit** as well.

## H Online Protocol

We present the online phase of the MPC scheme in Figure 27 and Figure 28.

$\Pi_{\mathsf{Online}}$ **(Part 1)**

The protocol runs among parties $\mathcal{P}$ which evaluate the circuit $\mathcal{C}$ over $\mathbb{F}$, and the auditor $\mathcal{P}_{\mathrm{A}}$. All parties have access to the bulletin board $\mathcal{F}_{\mathsf{GBB}}$ and the offline functionality $\mathcal{F}_{\mathsf{Offline}}$. The communication is synchronous. When a party does not send a message, $(\mathsf{Abort}, \mathsf{sid})$ is send through $\mathcal{F}_{\mathsf{GBB}}$ and the protocol is aborted. $\mathcal{F}_{\mathsf{Offline}}$ uses challenge space $\mathcal{T} = \mathbb{F}_p^*$.

**Initialize** Given the circuit $\mathcal{C}$ let $\mathsf{in}$ be the number of input values and $\mathsf{m}$ for the number of multiplication gates. Let $\mathtt{mp}_{\mathrm{val}}$ be a map that is initially empty.

1. $\mathcal{P}_1$ sends $\mathcal{C}$ to $\mathcal{F}_{\mathsf{GBB}}$. If the circuit is incorrect, then abort.
2. All parties send $(\mathsf{Init}, \mathsf{sid}, \mathsf{in}, 2\mathsf{m})$ to $\mathcal{F}_{\mathsf{Offline}}$.
3. All parties then invoke $(\mathsf{Input}, \mathsf{sid})$ and $(\mathsf{Triples}, \mathsf{sid})$. Each party stores all shares it obtains in $\mathtt{mp}_{\mathrm{val}}$.
4. Use **Verify Triples** to verify that the triples are correct.

**Input** $\mathcal{P}_i$ wants to share input $x \in \mathbb{F}$ under identifier $id_x$.

1. Let $id_r$ be the first unused id in $\mathtt{mp}_{\mathrm{val}}$.
2. Each party sends $(\mathsf{Open}, \mathsf{sid}, id_r, 0, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{Offline}}$.
3. Upon obtaining $(\mathsf{Open}, \mathsf{sid}, i, r)$ from $\mathcal{F}_{\mathsf{Offline}}$, $\mathcal{P}_i$ computes $\delta = x - r$ and sends $(\mathsf{Broadcast}, \mathcal{P}_i, \delta)$ to $\mathcal{F}_{\mathsf{GBB}}$.
4. Upon seeing $\delta$ on $\mathcal{F}_{\mathsf{GBB}}$, each party sends $(\mathsf{linear}, \mathsf{sid}, id_r, id_x, 1, \delta)$ to $\mathcal{F}_{\mathsf{Offline}}$.

**Procedure Verify Triples**

1. Let $\tau$ be the challenge returned by $\mathcal{F}_{\mathsf{Offline}}$ when generating triples.
2. Each of the parties compute $\rho = \tau \cdot x + a$ and $\sigma = y + b$ using linear operations in $\mathcal{F}_{\mathsf{Offline}}$.
3. The parties then reconstruct $\rho$ and $\sigma$ using $\mathcal{F}_{\mathsf{Offline}}$.
4. $\mathcal{P}_1$ broadcasts $\rho, \sigma, \tau$. If the value differs from the opened value, abort.
5. All parties compute $v = \tau \cdot z - c + \sigma \cdot a + \rho \cdot b - \rho \cdot \sigma$ using linear operations in $\mathcal{F}_{\mathsf{Offline}}$.
6. Open $v$ by sending $(\mathsf{Open}, \mathsf{sid}, id_v, 1, 0)$ to $\mathcal{F}_{\mathsf{Offline}}$. If $v \neq 0$, abort.

Fig. 27: Online protocol $\Pi_{\mathsf{Online}}$ that securely realizes $\mathcal{F}_{\mathsf{Online}}$ (Input and Computation).

### H.1 Security of the online protocol

*Proof.* We prove Theorem 4 by providing a simulator $\mathcal{S}_{\mathsf{OnHonP}}$( Figure 29) when at least one party is honest and $\mathcal{S}_{\mathsf{OnFull}}$( Figure 30) for the fully malicious case. The adversary can cheat by either introducing an error into the multiplication triples such that $a \cdot b + \Delta = c$ with $\Delta \neq 0$ or adding an error to a public, private or delayed opening. In both settings the simulator will first initialize $\mathcal{F}_{\mathsf{Online}}$, start a local copy of $\Pi_{\mathsf{Online}}$ and initialize $\mathcal{F}_{\mathsf{Offline}}$ with which all dishonest parties communicate. The simulation of **Linear** and **Multiply** is straight forward in either setting. The main difference lies in the point in time the simulator outputs that $\mathcal{A}$ cheated.

$\Pi_{\mathsf{Online}}$ **(Part 2)**

**Compute** When **Init** and **Input** have been executed before and all input gates of $\mathcal{C}$ have inputs assigned, all parties do the following for each gate in $\mathcal{C}$ with inputs $id_x, id_y$ in $\mathtt{mp}_{\mathrm{val}}$ and where the output $id_z$ has not yet been computed:

**Linear** Evaluate the gate using **Linear** in $\mathcal{F}_{\mathsf{Offline}}$. Then store $id_z$ in $\mathtt{mp}_{\mathrm{val}}$.

**Multiply**

1. The parties find an unused triple $id_a, id_b, id_c$.
2. The parties use **Linear** of $\mathcal{F}_{\mathsf{Offline}}$ to compute $\alpha = x - a$, $\beta = y - b$. Denote the outputs as $id_\alpha, id_\beta$.
3. Every $\mathcal{P}_i \in \mathcal{P}$ sends $(\mathsf{Open}, \mathsf{sid}, id_\alpha, 1, i)$ and $(\mathsf{Open}, \mathsf{sid}, id_\beta, 1, i)$ to $\mathcal{F}_{\mathsf{Offline}}$.
4. $\mathcal{P}_1$ sends $\alpha, \beta$ to $\mathcal{F}_{\mathsf{GBB}}$. If the values are incorrect, then abort.
5. Upon having obtained $\alpha, \beta$ from $\mathcal{F}_{\mathsf{Offline}}$, each party computes $id_z$ on $\mathcal{F}_{\mathsf{Offline}}$ where $z = \alpha \cdot b + \beta \cdot a - \alpha \cdot \beta + c$ using **Linear**.
6. Mark $id_a, id_b, id_c$ as used.

**Output** After all gates are evaluated, the parties open the output as follows:

1. Let $id_y$ be the ID of the output. All parties send $(\mathsf{Open}, \mathsf{sid}, id_y, 2, 0)$ to $\mathcal{F}_{\mathsf{Offline}}$.
2. Every party sends $(\mathsf{Release}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Offline}}$.
3. Let $L_o$ be the list of all opened $\alpha, \beta, \rho, \sigma, v$ during the computation and triple verification. Send $(\mathsf{Verify}, \mathsf{sid}, L_o)$ to $\mathcal{F}_{\mathsf{Offline}}$.
4. For $\mathcal{P}_i$ let $L_i$ list all IDs opened privately to $\mathcal{P}_i$ during **Input**. Send $(\mathsf{Verify}, \mathsf{sid}, L_i)$ to $\mathcal{F}_{\mathsf{Offline}}$.
5. If **Verify** outputs $\mathsf{reject}$ then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{GBB}}$ and abort.
6. If no party aborts then all parties send $(\mathsf{Release}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Offline}}$ and obtain $y$.
7. Send $(\mathsf{Verify}, \mathsf{sid}, id_y)$ to $\mathcal{F}_{\mathsf{Offline}}$. If $\mathcal{F}_{\mathsf{Offline}}$ responds with *Accept* then output $y$. Otherwise send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{GBB}}$ and abort.

**Audit** Upon input $(\mathsf{Audit}, \mathsf{sid}, y')$ to the auditor $\mathcal{P}_{\mathrm{A}}$, it does the following:

1. Check if any party broadcast $(\mathsf{Abort}, \mathsf{sid})$ on $\mathcal{F}_{\mathsf{GBB}}$. If so, then output $(\mathsf{NoAuditPossible}, \mathsf{sid})$.
2. Send $(\mathsf{Challenge}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Offline}}$ to obtain $\tau$.
3. $\mathcal{P}_{\mathrm{A}}$ sends $(\mathsf{Verify}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{GBB}}$ to retrieve the list of all opened messages denoted by $\mathtt{li}_{\mathrm{m}}$ which opened all $\alpha, \beta, \rho, \sigma, v$ and the output $y$. $\mathcal{P}_{\mathrm{A}}$ changes $\mathtt{li}_{\mathrm{m}}$ so that it also contains all the applied linear functions $f_l$ which can be reconstructed using $\tau$.
4. Let $id_y$ be the output and $y$ be the opened value on $\mathcal{F}_{\mathsf{GBB}}$. If $y \neq y'$ then output $(\mathsf{Reject}, \mathsf{sid})$.
5. $\mathcal{P}_{\mathrm{A}}$ sends $(\mathsf{Audit}, \mathsf{sid}, \mathtt{li}_{\mathrm{m}})$ to $\mathcal{F}_{\mathsf{Offline}}$. If $\mathcal{F}_{\mathsf{Offline}}$ returns $\mathsf{reject}$, output $(\mathsf{Reject}, \mathsf{sid})$. Otherwise output $(\mathsf{Accept}, \mathsf{sid})$.

Fig. 28: Online protocol $\Pi_{\mathsf{Online}}$ that securely realizes $\mathcal{F}_{\mathsf{Online}}$

*At least one honest party.* When at least one party is honest, $\mathcal{S}$ has to simulate these parties in the local copy of $\Pi_{\mathsf{Online}}$. When $\mathcal{A}$ introduces an additive error to any of the triples set $\mathtt{fl}_{\mathrm{cheated}} = \top$ but only abort the execution when $v$ opens to

another value than 0 in the **Verify Triple** procedure. Additionally, when using any opening and $\mathcal{A}$ adds an error to the opened value $\mathtt{fl}_{\text{cheated}}$ will be set to $\top$. During **Output** a delayed opening of $y^*$ is performed followed by a verification of all previously opened values, including their linear computations. Each party also includes the values that were privately opened to them in the verification process. Any honest party would abort the execution when **Verify** in $\mathcal{F}_{\mathsf{Offline}}$ returns reject, so $\mathcal{S}$ aborts if $\mathtt{fl}_{\text{cheated}} = \top$ which includes all of these cases and any other case of cheating. After $y^*$ is released the parties by reprogramming $\mathcal{F}_{\mathsf{Offline}}$ they perform another verification, similarly honest parties would abort the execution based on the answer of $\mathcal{F}_{\mathsf{Offline}}$. $\mathcal{S}$ will forward $y^*$ to $\mathcal{F}_{\mathsf{Online}}$ which will abort if $y^* \notin \{y, \perp\}$, where $y$ is the correct value computed by $\mathcal{F}_{\mathsf{Online}}$. **Audit** is straight forward in this setting as the protocol output is identical to the $\mathcal{F}_{\mathsf{Online}}$ output. $\mathcal{S}$ will always abort the execution while honest parties in a protocol will not abort the execution in the case of a cheating $\mathcal{A}$, when $\mathcal{A}$ wins in the triple check despite triples being inconsistent. We discuss the probability for this scenario after we present the fully malicious setting.

*Fully malicious setting.* When no party is honest there is not much simulation to be done, except of $\mathcal{F}_{\mathsf{Offline}}$ towards the corrupted parties. Moreover, $\mathcal{S}$ will not reveal cheating during **Output**, as no malicious party would abort the computation even if $\mathcal{F}_{\mathsf{Offline}}$ returns reject or $v \neq 0$ (unless they explicitly abort). Only when $\mathcal{P}_{\text{A}}$ queries **Audit** the protocol should reject in the case of malicious behavior. $\mathcal{F}_{\mathsf{Online}}$ will always output $(\mathsf{Reject}, \mathsf{sid}, y)$ if $\mathcal{A}$ cheated, but the protocol will falsely accept $y$ in the same case that honest parties will falsely accept the computation in the previously discussed case.

*False triple check.* The value $\tau$ is generated by $\mathcal{F}_{\mathsf{Offline}}$ and can be any value in $\mathbb{F} \setminus 0$. Any new choice of triples changes the value of $\tau$ and $\mathcal{A}$ can test $\tau$ when triples are sent during **Triples**. If the adversary is not satisfied with $\tau$ (i.e. the test would not pass), $\mathcal{A}$ can send reject and try new inputs, or reset the functionality by sending reset if all parties are corrupted. It is important that all triple shares of the adversary must be chosen before the adversary learns $\tau$. For each $\tau$ the probability of a false accepting **Triple Verification** is $\frac{1}{|\mathbb{F}|-1}$ as described and proven in [28]. Hence the total distinguishing advantage for any environment $\mathcal{Z}$ is $\frac{Q}{|\mathbb{F}|-1}$, where $Q$ equals the number of reject or reset messages that the adversary can send.

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

In the following we denote $\mathtt{H}$ as honest and $\mathtt{C}$ as corrupt parties. Initially set $\mathtt{fl}_{\text{cheated}} = \bot$. If at any point a party broadcasts $(\mathsf{Abort}, \mathsf{sid})$ then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Online}}$.

**Initialize**

1. Upon observing the broadcast of $\mathcal{C}$ and if no party aborts, send $(\mathsf{Init}, \mathsf{sid}, \mathcal{C})$ to $\mathcal{F}_{\mathsf{Online}}$.
2. Start a local instance of $\Pi_{\mathsf{Online}}$ with which all $\mathcal{P} \in \mathtt{C}$ communicate. Simulate honest parties in $\mathtt{H}$.
3. Initialize a copy of $\mathcal{F}_{\mathsf{Offline}}$ by running all steps of **Initialize**. All parties communicate with the instance of $\mathcal{F}_{\mathsf{Offline}}$ through the simulator.
4. Follow the steps of **Initialize** in $\Pi_{\mathsf{Online}}$. When $\mathcal{A}$ inputs any $\delta_a, \delta_b$ or $\Delta_c$ that is $\neq 0$ to $\mathcal{F}_{\mathsf{Offline}}$ during **Triple**, set $\mathtt{fl}_{\text{cheated}} = \top$.
5. When the opening of $v$ during the **Verify Triples** procedure is $v \neq 0$, abort the execution.

**Input**

1. For every $\mathcal{P}_i \in \mathtt{H}$, simulate $\Pi_{\mathsf{Online}}$ honestly using a dummy input upon receiving $(\mathsf{Input}, \mathsf{sid}, i, \mathsf{id}_x)$ from $\mathcal{F}_{\mathsf{Online}}$. If any party in $\mathtt{C}$ inputs $\Delta \neq 0$ to $\mathcal{F}_{\mathsf{Offline}}$ during **Open**, set $\mathtt{fl}_{\text{cheated}} = \top$.
2. For any party $\mathcal{P}_i \in \mathtt{C}$ wait until $\mathcal{P}_i$ sends $\delta$ through $\mathcal{F}_{\mathsf{GBB}}$. Compute $x = \delta + r$ and send $(\mathsf{Input}, \mathsf{sid}, i, \mathsf{id}_x, x)$ to $\mathcal{F}_{\mathsf{Online}}$.

**Compute** When **Initialize** and **Input** have been executed before, evaluate $\mathcal{C}$ gate per gate. For **Linear** and **Multiply** follow the steps in $\Pi_{\mathsf{Online}}$.

**Output**

1. Simulate the delayed open during **Output**. The adversary inputs $\Delta$ into $\mathcal{F}_{\mathsf{Offline}}$.
2. Run the first $(\mathsf{Release}, \mathsf{sid})$ with $\mathcal{F}_{\mathsf{Offline}}$. If $\mathtt{fl}_{\text{cheated}} = \top$ then abort.
3. Otherwise send $(\mathsf{Compute}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Online}}$ and wait for the output $y$.
4. Wait for parties to send the second $(\mathsf{Release}, \mathsf{sid})$ and let $\mathcal{F}_{\mathsf{Offline}}$ output $y^* = y + \Delta$. If $\Delta \neq 0$ let $\mathcal{F}_{\mathsf{Offline}}$ output $\mathsf{reject}$ during **Verify** of $y^*$ and $\mathsf{accept}$ if $\Delta = 0$.
5. Send $(\mathsf{Output}, \mathsf{sid}, y^*)$ to $\mathcal{F}_{\mathsf{Online}}$.

Fig. 29: Simulator for the online phase where at least one party is honest.

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

In the following all parties are corrupt. Initially set $\mathtt{fl}_{\text{cheated}} = \bot$. The simulator waits until the computation is complete and all messages are send to $\mathcal{F}_{\mathsf{GBB}}$ before the simulation begins. If at any point a party broadcasts $(\mathsf{Abort}, \mathsf{sid})$ then send $(\mathsf{Abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Online}}$.

**Initialize**

1. Initialize a copy of $\mathcal{F}_{\mathsf{Offline}}$ with which the parties communicate.
2. Upon observing the circuit $\mathcal{C}$ being broadcast, send $(\mathsf{Init}, \mathsf{sid}, \mathcal{C})$ to $\mathcal{F}_{\mathsf{Online}}$.
3. Emulate an instance of $\mathcal{F}_{\mathsf{Offline}}$ honestly.
4. If $\mathcal{A}$ sends a malformed triple to $\mathcal{F}_{\mathsf{Offline}}$, set $\mathtt{fl}_{\text{cheated}} = \top$.

**Input** Wait for $\mathcal{P}_i$ to broadcast $\delta$. Input $x = \delta + r$ to $\mathcal{F}_{\mathsf{Online}}$, where $r$ is the value opened through $\mathcal{F}_{\mathsf{Offline}}$.

**Compute** If any of the openings during a multiplication is incorrect, set $\mathtt{fl}_{\text{cheated}} = \top$.

**Output**

1. Upon having observed all openings of $\mathcal{F}_{\mathsf{GBB}}$ and if $(\mathsf{Release}, \mathsf{sid})$ has been used as in the protocol, send $(\mathsf{Compute}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Online}}$ and obtain $y$. Let $\Delta$ be the value sent through **Open** when releasing the output.
2. Set $y^* = y + \Delta$ and send $(\mathsf{Output}, \mathsf{sid}, y^*)$ to $\mathcal{F}_{\mathsf{Online}}$. If $y^* \notin \{y, \bot\}$, set $\mathtt{fl}_{\text{cheated}} = \top$.
3. If $\mathtt{fl}_{\text{cheated}} = \top$ then let **Audit** in $\mathcal{F}_{\mathsf{Offline}}$ always fail during audit of the output $y^*$.

Fig. 30: Simulator for the online phase in the fully malicious setting.