Committed Vector Oblivious Linear Evaluation and Its Applications

Yunqing Sun* Hanlin Liu* Kang Yang† Yu Yu‡ Xiao Wang*
Chenkai Weng§

Abstract

We introduce the notion of committed vector oblivious linear evaluation (C-VOLE), which allows a party holding a pre-committed vector to generate VOLE correlations with multiple parties on the committed value. It is a unifying tool that can be found useful in zero-knowledge proofs (ZKPs) of committed values, actively secure multi-party computation, private set intersection (PSI), etc.

To achieve the best efficiency, we design a tailored commitment scheme and matching C-VOLE protocols, both based on the learning parity with noise assumption. In particular, exploiting the structures of the carefully designed LPN-based commitment minimizes the cost of ensuring consistency between the committed vector and VOLE correlation. As a result, we achieve an 28× improvement over the protocol proposed in prior work (Usenix 2021) that uses ZKP to prove the correct opening of the commitment. We also apply C-VOLE to design a PSI protocol that allows one server to run PSI repeatedly with multiple clients while ensuring that the same set is used across all executions. Compared with the state-of-the-art PSI (CCS 2024) with similar security requirements, our protocol reduces the communication overhead by a factor of 35×.

1 Introduction

Vector oblivious linear evaluation (VOLE) is a type of protocol that sets up linear correlations between two parties. It can be viewed as an arithmetic extension of the correlated oblivious transfer [NNOB12]. By running a VOLE protocol between $\mathsf{P_A}$, who has an input vector $\boldsymbol{x} \in \mathbb{F}^m$, and $\mathsf{P_B}$, $\mathsf{P_A}$ would obtain a uniform vector $\mathsf{M}[\boldsymbol{x}] \in \mathbb{F}^m$ as output, and $\mathsf{P_B}$ would obtain $(\Delta, \mathsf{K}[\boldsymbol{x}]) \in \mathbb{F} \times \mathbb{F}^m$ such that they satisfy the correlation $\mathsf{M}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta$. VOLE correlation can be viewed as an information-theoretic message authentication code (IT-MAC) on $\mathsf{P_A}$'s input, hence serving as a commitment or materials to construct more useful correlations for MPC like Beaver triples. Therefore, extensive practical applications in zero-knowledge proofs (ZKPs) [BMRS21, DIO21, WYKW21], multi-party computation [BDOZ11, DILO22a, RS22], private set intersection (PSI) [RR22a, RS21], etc, have been found recently.

In many VOLE-based protocols, it is often desirable to allow $\mathsf{P_A}$ to 1) first non-interactively commit to its input, which can be distributed to all parties easily, and later 2) run VOLE with other parties while ensuring that $\mathsf{P_A}$ uses the committed input to the VOLE protocol. Such a capability enforces $\mathsf{P_A}$ to use the same input across multiple executions of VOLE, potentially with

*Northwestern University, {yunqing.sun,hanlin.liu,wangxiao}@northwestern.edu
†State Key Laboratory of Cryptology, yangk@sklc.org
‡Shanghai Jiao Tong University and Shanghai Qi Zhi Institute, yuyu@yuyu.hk
§Arizona State University, Chenkai.Weng@asu.edu

different parties. Moreover, the commitment could be pre-distributed to parties offline using cheaper mechanisms like content distribution networks. In more detail:

1. VOLE-based ZKP [WYKW21, BMRS21, DIO21] is a recent paradigm for constructing lightweight ZKP protocols. However, they are designated verifier ZK and do not provide a cheap mechanism if the statement needs to use a consistent witness across different ZKPs. A recent work [WYX+21] explored proving the decommitment phase of a symmetric-key-based commitment; however, its performance is fairly limited due to the large size of SHA-256 and AES. Other works [AGR+16, GKR+21, AAB+20, GKS23] have explored designing ZK-friendly hash functions, but the cost is high as the underlying commitment schemes always require non-black-box use.

2. Among existing dishonest-majority MPC protocols, the most practical way to achieve malicious security is to use IT-MACs to authenticate secret shares [BDOZ11, DPSZ12, WRK17, RS22]. These protocols often require running pair-wise VOLE across all parties so that even if all but one parties are corrupted, the underlying secret sharing is still authenticated. However, a critical issue is ensuring that every party uses the same input vector for all VOLE executions with every other party. To achieve that goal, existing MPC protocols require a joint consistency check among all the parties, which often involves significant computation and communication.

3. Crowd PSI refers to a specific type of PSI protocol involving a server and many clients. The server has a large set and repeatedly runs a PSI protocol with many individual clients, using the same set across different PSI executions. It is useful when in services where many clients need to privately look up some private information on the server side, like password checkup or private contact discovery. Prior work [SKR+24] in this setting with malicious security requires a public-key-based pseudorandom function (PRF) and its oblivious evaluation for every element. To adapt existing VOLE-based PSI protocols [RS21, RR22a], the most performant PSI protocols right now, to work in this setting, one would need to ensure that the same input vector from the server is used for all VOLE executions, which in turn ensures consistency in the server's set across all executions.

To summarize, the above examples either require P_A's VOLE input vector $\boldsymbol{x}$ to be publicly committed beforehand or require P_A's VOLE input vector $\boldsymbol{x}$ to remain the same among multiple VOLE applications with different parties, which can also be ensured if all parties have the same commitment of $\boldsymbol{x}$. This paper provides a unified approach to solve all these issues efficiently in a modular way. To this end, we propose *committed-VOLE (C-VOLE)*, which allows the VOLE sender's input to be pre-committed non-interactively.

1.1 Our Contribution

In this work, we propose concretely efficient C-VOLE protocols based on the assumption of learning parity with noise (LPN). Then, we show how C-VOLE can be used in multiple important protocols and how our new protocols can enhance the efficiency of these applications.

LPN-based C-VOLE protocol. We propose a C-VOLE protocol based on the LPN assumption. Prior work [WYX+21] proposed a protocol using a commitment scheme based on random oracle and AES and proved the correct decommitment, which involves computing SHA256 and AES within ZK. We design an LPN-based commitment scheme specifically for C-VOLE to achieve the best possible efficiency. Then, we exploit the fact that this commitment scheme and the state-of-the-art VOLE protocols share a similar structure, i.e., the LPN-like structure, to design a zero-knowledge

proof of decommitment with very small communication. As a result, our C-VOLE protocol achieves significantly better online communication efficiency than the corresponding VOLE protocol, while requiring 2× the online running time. We show how our protocol can be based on either dual-LPN-based VOLE or primal-LPN-based VOLE.

C-VOLE applications and their performance. We provided some concrete case studies of using C-VOLE in prior applications and how it can improve the efficiency of previous works. Additionally, we implement our C-VOLE protocol and benchmark its practical performance.

For our C-VOLE protocol, the offline commitment is nearly the same size as the data being committed. The online communication is almost constant, with a running time of about 900 ns and 420 ns per element in a LAN setting using 4 and 16 threads, respectively, for a vector of length 2^{26}. Note that running VOLE with a chosen input requires essentially the same amount of communication as the input size to "fix" a random correlation to the chosen input. Therefore, ours communication overhead is near optimal while most communication is offline. We also investigate the protocol change and the concrete efficiency when applying our C-VOLE protocol in various applications.

1. C-VOLE directly enables ZK proof on publicly committed values. Compared to Mystique, we observe a 28× improvement in running time and 7× improvement in communication cost.

2. Regarding the multi-party VOLE in generic MPC protocols, C-VOLE avoids the final consistency check, which contains heavy peer-to-peer and broadcast communications. In the context of multi-party VOLE in dishonest-majority MPC and multi-verifier ZKPs [RS22, HSS20, WRK17, EPSW24], C-VOLE similarly avoids the costly final consistency check, significantly reducing the round complexity.

3. By adapting C-VOLE to the VOLE-PSI protocol, we can convert state-of-the-art, actively secure one-shot PSI protocol to the crowd setting without assuming the server's honest behavior. The C-VOLE PSI protocol requires around 750 ns per server element for computing one-time commitment and around 900 ns per server element for the online interaction with each client, given the server set size of 2^{24}. It reduces online communication overhead by 35× compared to the state-of-the-art PSI protocol in the crowd setting [SKR+24].

Other related notions. Note that a different notion of committing VOLE is recently defined in [CDKs24] to construct MPC with an input-revealing identifiable abort guarantee. Their focus is a VOLE variant where parties inputs and outputs are committed so that they can later be opened. Therefore, their notion does not provide input consistency between different executions of VOLE. Thus, their notion is more similar to committing OT notation proposed by Jawurek et al. [JKO13].

2 Technical Overview

In this section, we provide some intuition of our constructions.

Prior VOLE constructions Recall that a *random* VOLE protocol outputs $(\boldsymbol{u}, \mathsf{M}[\boldsymbol{u}]) \in \mathbb{F}^m \times \mathbb{F}^m$ to $\mathsf{P_A}$ and $(\Delta, \mathsf{K}[\boldsymbol{u}]) \in \mathbb{F} \times \mathbb{F}^m$ to $\mathsf{P_B}$, such that all values are uniform under the constraint that

$$\mathsf{M}[\boldsymbol{u}] = \mathsf{K}[\boldsymbol{u}] + \boldsymbol{u} \cdot \Delta.$$

When two parties hold VOLE correlations for vector $\boldsymbol{u}$ like the above, we also write $[\![\boldsymbol{u}]\!]$ for simplicity. All of the state-of-the-art VOLE protocols [WYKW21, BCG+22, RRT23] compute such

random correlation because this kind of correlations can be easily compressed to achieve sublinear communication and they can be easily derandomized to chosen input. More specifically, if P_A wants to correct $\boldsymbol{u}$ to a designated vector $\boldsymbol{x}$, it simply sends the difference $\boldsymbol{d} = \boldsymbol{u} - \boldsymbol{x}$ to P_B and defines $\mathsf{M}[\boldsymbol{x}] = \mathsf{M}[\boldsymbol{u}]$; P_B can define $\mathsf{K}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{u}] + \boldsymbol{d} \cdot \Delta$. Now these new values form a correlation

$$\mathsf{M}[\boldsymbol{x}] = \mathsf{M}[\boldsymbol{u}] = \mathsf{K}[\boldsymbol{u}] + \boldsymbol{u} \cdot \Delta = \mathsf{K}[\boldsymbol{u}] + (\boldsymbol{u} - \boldsymbol{x}) \cdot \Delta + \boldsymbol{x} \cdot \Delta = \mathsf{K}[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta.$$

To efficiently construct random VOLE, state-of-the-art protocols follow the pseudorandom correlation generator (PCG) framework [BCGI18]. For example, the dual version of the learning parity with noise (LPN) assumption states that

$$(\mathbf{H}, \boldsymbol{u}) \text{ is computationally indistinguishable from } (\mathbf{H}, \boldsymbol{e}_u\mathbf{H}),$$

where $\mathbf{H} \in \mathbb{F}^{cm \times m}$ is a public random but structured matrix and $\boldsymbol{e}_u \in \mathbb{F}^{cm}$ is a sparse vector. To allow more efficient constructions, state-of-the-art protocols also assume that $\boldsymbol{e}_u$ is regular, meaning that it partition $\boldsymbol{e}_u$ into w number of shorter vectors $(\boldsymbol{e}_u^1, \ldots, \boldsymbol{e}_u^w)$ each of length 2^L such that each shorter vector is non-zero only at one random location. With this assumption, one can construct an efficient VOLE by first constructing "simple" VOLE on each $\boldsymbol{e}_u^i$'s to obtain $[\![\boldsymbol{e}_u^i]\!]$, following [BCG+19, BCGI18, BCG+17] with cost $O(L\kappa)$, where κ is the computational security parameter. Then, parties can assemble $[\![\boldsymbol{e}_u]\!] = [\![\boldsymbol{e}_u^1]\!] \| \ldots \| [\![\boldsymbol{e}_u^w]\!]$ locally and further compute $[\![\boldsymbol{e}_u\mathbf{H}]\!] = [\![\boldsymbol{e}_u]\!] \cdot \mathbf{H}$ due to linear homomorphism. As a result the total communication cost is sublinear. However, when $\mathsf{P_A}$ interacts with different parties, there is no guarantee that they will use the same randomness and hence no guarantee about the consistency of $\mathsf{P_A}$'s input.

Our main contribution is to design a customized commitment scheme and proof of consistency to force $\mathsf{P_A}$ using the same value with every party who holds the commitment.

Blueprint of our protocol. From a high-level view, our protocol functions as follows. First $\mathsf{P_A}$, with an input vector $\boldsymbol{x} \in \mathbb{F}^m$, can locally compute a commitment of size roughly $m \cdot \log|\mathbb{F}|$ bits. This commitment is distributed to all relevant parties by various means, such as a broadcasting protocol or public bulletin board, and its genuineness can be verified with a signature, assuming PKI exists. Because the same commitment is distributed to all parties, it can enjoy network caching at a reduced cost. Then $\mathsf{P_A}$ can interact with each individual party acting as $\mathsf{P_B}$ in VOLE and execute the online phase of the C-VOLE protocol such that they obtain VOLE correlation about $\boldsymbol{x}$ while the protocol ensures that $\mathsf{P_A}$ must use $\boldsymbol{x}$ as its input (otherwise the protocol would abort). The commitment is used for $\mathsf{P_A}$ to correct VOLE of random vector to its input $\boldsymbol{x}$. Different parties acting as $\mathsf{P_B}$ all know that $\mathsf{P_A}$ uses the same input under the pre-distributed commitment.

At the core of the protocol is the online phase, where a consistency check is needed to show that the committed $\boldsymbol{x}$ is consistent with the $\boldsymbol{x}$ in VOLE. Assuming a commitment scheme Com and a pseudorandom generator (PRG) PRG, one can convert the problem to ensuring the consistency of a seed as follows. The commitment to $\boldsymbol{x}$ is now $\mathsf{com}_{\boldsymbol{x}} = (\mathrm{PRG}(\boldsymbol{r}) - \boldsymbol{x}, \mathsf{Com}(\boldsymbol{r}))$. During the online phase, two parties obtain the VOLE correlation for $\boldsymbol{r}$ and $\boldsymbol{x}$; then, they can prove using VOLE-ZK that $\boldsymbol{r}$ satisfies two properties. First, when fed to the PRG, it yields a vector consistent with the first value in the commitment $\mathsf{com}_{\boldsymbol{x}}$, i.e.,

$$\mathrm{PRG}([\![\boldsymbol{r}]\!]) - [\![\boldsymbol{x}]\!] = \mathsf{com}_{\boldsymbol{x}}[0].$$

Second, $[\![\boldsymbol{r}]\!]$ is consistent with $\mathsf{Com}(\boldsymbol{r})$. Parties execute a proof of decommitment to obtain $[\![\mathsf{decom}]\!] = \mathsf{Open}(\mathsf{com}_{\boldsymbol{x}}[1])$. Then, $\mathsf{P_A}$ proves that

$$[\![\mathsf{decom}]\!] = [\![\boldsymbol{r}]\!].$$

Both are statements relevant to VOLE values; thus, VOLE-based ZK only requires one extra message from $\mathsf{P_A}$ but the communication complexity is linear to the circuit size of PRG and $\mathsf{Open}(\cdot)$. The rest of this section discusses how to construct PRG and Com so that proving the above two statements using VOLE-based ZK involves almost no communication.

Commitment based on LPN assumption. Since state-of-the-art VOLE uses LPN assumptions to generate pseudorandom bits, our goal is to design the commitment for it. Rigorously, the underlying PRG can be defined as $\mathrm{PRG} \in \{0,1\}^{wL} \times \mathbb{F}^w \to \{0,1\}^m$:

$$\mathrm{PRG}(k_1, \ldots, k_w; a_1, \ldots, a_w) = \mathsf{Unary}(k_1, a_1) \| \ldots \| \mathsf{Unary}(k_w, a_w) \cdot \mathbf{H},$$

where $\mathbf{H} \in \mathbb{F}^{w2^L \times m}$ is a public matrix and $\mathsf{Unary}(k_i, a_i)$ is defined as an all zero vector with a_i at position k_i. The seed of this PRG consists of wL bits and w elements in $\mathbb{F}$, while the output is a vector of m elements. In practice, to maintain 128-bit computational security, we set $w = 224$, $L = 23$, and $m = 2^{28}$, which means the output is much larger than the input, hence a good PRG.

The next step is to design a commitment scheme for $\boldsymbol{k} = (k_1, \ldots, k_w)$ and $\boldsymbol{a} = (a_1, \ldots, a_w)$. We propose to use an LPN-based commitment [JKPT12] due to its linearity. For example, to commit the vector $\boldsymbol{a}$, one can compute

$$\boldsymbol{a} \cdot \mathbf{H}_1 + \hat{\boldsymbol{r}} \cdot \mathbf{H}_2 + \hat{\boldsymbol{e}},$$

where $\mathbf{H}_1$ and $\mathbf{H}_2$ are two public matrices, $\hat{\boldsymbol{r}}$ is a uniform vector, and $\hat{\boldsymbol{e}}$ is a sparse vector, all with appropriate length. It can be shown that when picking the dimension of the two matrices and the sparsity of $\hat{\boldsymbol{e}}$ appropriately, $\hat{\boldsymbol{r}} \cdot \mathbf{H}_2 + \hat{\boldsymbol{e}}$ is pseudorandom and thus provides hiding, while it is still difficult to find two pairs of $(\boldsymbol{a}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{e}})$ and $(\boldsymbol{a}', \hat{\boldsymbol{r}}', \hat{\boldsymbol{e}}')$ that lead to the same value, hence binding.

Proving the consistency is fairly cheap: $\mathsf{P_A}$ and $\mathsf{P_B}$ can first run the VOLE protocol with $\mathsf{P_A}$ providing $\boldsymbol{k}$ and $\boldsymbol{a}$. As part of the VOLE protocol, two parties would obtain VOLE on values $\boldsymbol{e}_u^i = \mathsf{Unary}(k_i, a_i)$, namely $[\![\boldsymbol{e}_u^i]\!]$ for each $i \in [w]$. One can easily check the correct value of a_i by checking $a_i = \sum_{j \in [2^L]} \boldsymbol{e}_u^i[j]$, which is free. Checking the sparsity of $\boldsymbol{e}_u^i$ is no longer free but can still be done with linear cost by running the binary-to-unary circuit in ZKP from $k_i \in \{0,1\}^L$ to a Boolean vector of length 2^L, which takes a total of $2m$ non-linear gates. The cost is orders of magnitude better than Mystique, which requires more than 50 non-linear gate per output due to a large AES circuit.

C-VOLE with sublinear online communication. The above protocol is much faster than prior work, but the online communication cost is still linear to the vector size. Note that we already send linear communication as part of the offline commitment, so we ideally want the online phase to be even smaller. This means the consistency check can only prove a circuit of size sublinear to the output size!

In the above protocol, the most expensive operation is proof of sparsity; it is required for the binding property. Our crucial observation is that the VOLE protocol already ensures that each $[\![\boldsymbol{e}_u^i]\!]$ is sparse; we need to prove the binary-to-unary circuit only to ensure that it is consistent with the committed randomness. To this end, we modify our commitment. Instead of committing to the seed of the LPN-PRG, we commit to some intermediate values in the PRG, i.e., the structured sparse vector, directly. More specifically, our commitment scheme no longer handles any input but only sparse vectors $\boldsymbol{e}_u$:

$$\boldsymbol{e}_u \cdot \mathbf{H}_1 + \boldsymbol{e}_r \cdot \mathbf{H}_2,$$

where $\mathbf{H}_1$ and $\mathbf{H}_2$ are two public matrices, and $\boldsymbol{e}_r$ is a sparse vector. We can show that it is computationally infeasible to obtain two $(\boldsymbol{e}_u, \boldsymbol{e}_r)$ and $(\boldsymbol{e}_u', \boldsymbol{e}_r')$ such that the above commitments are the same and that $\boldsymbol{e}_u'$ is also regular with specified sparsity. Note that because the output is much

shorter than $|\boldsymbol{e}_u|$, finding a collision is possible if the sparsity is not guaranteed by other means; but again, in our application, VOLE protocols already ensure it for free.

Now, our protocol is extremely efficient. Two parties obtain $[\![\boldsymbol{e}_u]\!]$ as part of the normal VOLE protocol. They just need to obtain $[\![\boldsymbol{e}_r]\!]$ and check if the $[\![\boldsymbol{e}_u]\!]\cdot\mathbf{H}_1+[\![\boldsymbol{e}_r]\!]\cdot\mathbf{H}_2$ matches the commitment. We show that $|\boldsymbol{e}_r| \ll |\boldsymbol{e}_u|$, and thus, this extra communication cost is very small compared to the cost of the VOLE protocol. In addition to the VOLE protocol, the main computational cost is the evaluation of $[\![\boldsymbol{e}_u]\!]\cdot\mathbf{H}_1$, which is only a local computation. We provide the detailed description and analysis of LPN-based commitment in Section 4.1 and Section A.1

Applications to VOLE-ZK, MPC, and MVZK. C-VOLE brings immediate performance improvement when used in several recent VOLE-ZK and MPC protocols.

- In VOLE-based ZKP protocols, C-VOLE directly leads to a way to convert public commitments to VOLE-like authentication ready for VOLE-ZK. Compared to the previous work [WYX+21], it avoids proving large SHA-256 and AES circuits in ZKP and achieves 28× concrete improvement.
- For generic secure multi-party computation, it can be used to execute multi-party VOLE among all parties [RS22, HSS20, WRK17]. By using C-VOLE, they no longer require heavy peer-to-peer and broadcast communication in the final consistency check. MVZK with dishonest-majority verifiers [EPSW24] also relies on similar consistency checks, which can be avoided by C-VOLE.

We defer the details to Section 6.

Application to crowd PSI. In the crowd PSI setting, we have one server holding a large set X and a set of clients, each holding a small set Y_i, such that each pair want to obtain the intersection $X \cap Y_i$. State-of-the-art PSI protocols based on VOLE [RS21, RR22a] can provide malicious security but cannot ensure consistency of X. A prior work [SKR+24] works in this setting but uses public-key based Oblivious PRF (OPRF), which is highly expensive.

We observe that the only place where VOLE-PSI protocols use VOLE is for the server to feed its input set to the VOLE. The server's input set is encoded into a vector $\boldsymbol{p}$, such that $\mathsf{Decode}(\boldsymbol{p}, x) = H_1(x)$ for $x \in X$. The server identifies any client's set elements in the intersection using an OPRF defined by $[\![\boldsymbol{p}]\!]$. Specifically, the client encodes its set elements by $F(x) = H_2(\mathsf{Decode}(\mathsf{K}[\boldsymbol{p}], x) + \Delta \cdot H_1(x), x)$, where H_1 and H_2 are two hash functions, and sends the encodings to the server. The server encodes its set elements by $X' := \{H_2(\mathsf{Decode}(\mathsf{M}[\boldsymbol{p}], x), x) | x \in X\}$ and checks for matches with the client's encoding. The two encodings match if the client's set element is located in the intersection, which satisfies $\mathsf{Decode}(\boldsymbol{p}, x) = H_1(x)$. In this case, VOLE is utilized as a black-box component in the VOLE-PSI protocol. Compared to VOLE, C-VOLE enforces the consistency of $\boldsymbol{p}$ across different VOLE executions with other clients, ensuring consistent use of the server's set. Thus, we can prove that if we replace VOLE with C-VOLE, we can obtain a PSI protocol in the crowd setting.

As a result, we upgrade the state-of-the-art single-client PSI without the set-committing feature into a crowd PSI. The C-VOLE PSI requires around 900 ns per server element for online computation and less than 1.64 B per server element for online communication of set size 2^{20}, which is roughly 2× online time consuming but 13× better online communication overhead compared with VOLE-PSI. Compared with prior work [SKR+24] in the crowd setting, C-VOLE PSI reduces initialization time with a 8-fold improvement for set size 2^{20}. It also decreases online communication to 1.73 MB with clients having 2^{10} elements, reduced by a factor of 35×.

3 Preliminaries

3.1 Notation

Define statistical and computational security parameters λ and κ. We let $\mathsf{negl}(\cdot)$ denote a negligible function, and use log to denote logarithms in base 2. $\mathbb{F}$ denotes a finite field. We write $x \leftarrow \mathbb{F}$ to denote sampling x uniformly from $\mathbb{F}$. We write $[n] = \{1, \ldots, n\}$. We use bold upper-case letters like $\mathbf{A}$ for matrices. For $\mathbf{A} \in \mathbb{F}^{m \times n}$, m denotes the number of rows, and n denotes the number of columns. We use bold lower-case letters like $\boldsymbol{a}$ for row vectors, and let $\boldsymbol{a}[i]/a_i$ denote the ith component of $\boldsymbol{a}$ (with $\boldsymbol{a}[1]/a_1$ the first entry). $\boldsymbol{a}^T$ denotes the transposed vector of $\boldsymbol{a}$. We use $\|$ to represent the concatenation. Define a regular-w vector $\boldsymbol{e}$ as $\boldsymbol{e} = (\boldsymbol{e}^1 \| \ldots \| \boldsymbol{e}^w) \in \mathbb{F}^{w \cdot 2^L}$, where $\boldsymbol{e}^i \in \mathbb{F}^{2^L}$ is a unary vector. We also denote $\boldsymbol{e}$ as a multi-point vector and $\boldsymbol{e}^i$ as a single-point vector. We use $\mathsf{K}[\boldsymbol{a}]$ to represent $\mathsf{K}[a_1]||\cdots||\mathsf{K}[a_n]$, and let $\mathsf{M}[\boldsymbol{a}] = \mathsf{M}[a_1]||\cdots||\mathsf{M}[a_n]$.

3.2 Learning Parity with Noise

The Learning Parity with Noise (LPN) problem is defined as recovering a secret vector given a noisy system of linear equations. It has found many applications in cryptography, as it has strong security guarantees. The hardness definition of primal-LPN and dual-LPN [BFKL94] are presented in Definition 1 and Definition 2 respectively as follows:

Definition 1. *(Primal LPN) Let $\mathcal{C}$ be a probabilistic code generation algorithm such that $\mathcal{C}(N, k, \mathbb{F})$ outputs a matrix $\mathbf{A} \in \mathbb{F}^{k \times N}$. We say that the decisional primal-LPN(N, k, t) problem is (T, ε)-computationally hard for every probabilistic distinguisher $\mathcal{D}$ running in time T if*

$$|Pr[\mathcal{D}(\mathbf{A}, \mathbf{A} \cdot \boldsymbol{x} + \boldsymbol{e}) = 1] - Pr[\mathcal{D}(\mathbf{A}, \boldsymbol{u}) = 1]| \leq \varepsilon,$$

where $\boldsymbol{x} \leftarrow \mathbb{F}^N$, $\boldsymbol{u} \leftarrow \mathbb{F}^k$, and $\boldsymbol{e} \leftarrow \mathbb{F}^k$ with weight t.

Definition 2. *(Dual LPN) Let $\mathcal{C}^{\perp}$ be a probabilistic code generation algorithm such that $\mathcal{C}^{\perp}(N, k, \mathbb{F})$ outputs a matrix $\mathbf{H} \in \mathbb{F}^{k \times N}$. We say that the decisional dual-LPN(N, k, t) problem is (T, ε)-computationally hard for every probabilistic distinguisher $\mathcal{D}$ running in time T if*

$$|Pr[\mathcal{D}(\mathbf{H}, \mathbf{H} \cdot \boldsymbol{e}) = 1] - Pr[\mathcal{D}(\mathbf{H}, \boldsymbol{u}) = 1]| \leq \varepsilon,$$

where $\boldsymbol{e} \leftarrow \mathbb{F}^N$ with weight t, and $\boldsymbol{u} \leftarrow \mathbb{F}^k$.

3.3 Commitment Scheme

A commitment scheme is defined as a sender committing to a chosen value while keeping it hidden from others and opening the value later to let others verify it.

1. **ParamGen:** The sender samples public parameter k from a parameter space K and publish k to other parties.
2. **Commit:** The sender picks message $\boldsymbol{x} \leftarrow \mathbb{F}^m$ and computes $\{\mathsf{com}_{\boldsymbol{x}}, \mathsf{decom}_{\boldsymbol{x}}\} \leftarrow \mathsf{Com}(\boldsymbol{x})$, where $\mathsf{decom}_{\boldsymbol{x}} \in \mathbb{F}^*$. The sender sends $\mathsf{com}_{\boldsymbol{x}}$ to the receiver.
3. **Open:** The sender sends x and $\mathsf{decom}_{\boldsymbol{x}}$ to the receiver. The receiver computes $0/1 \leftarrow \mathsf{Open}(\mathsf{com}_{\boldsymbol{x}}, \mathsf{decom}_{\boldsymbol{x}}, \boldsymbol{x})$

A commitment scheme with computational hiding and statistical binding is defined as follows.

Functionality $\mathcal{F}_{\mathsf{C-VOLE}}$

Public Parameters: A public commitment of $\boldsymbol{x}$, $\mathsf{Hcom}_{\boldsymbol{x}}$. A hybrid commitment scheme $\mathsf{HCom}(\cdot)$ defined in Section 4.2.

1. Upon receiving $(\mathsf{init}, \boldsymbol{x}, \mathsf{Hdecom}_{\boldsymbol{x}})$ from P_A, and (init) from P_B, the functionality stores $(\boldsymbol{x}, m = |\boldsymbol{x}|)$ and checks whether $(\mathsf{Hcom}_{\boldsymbol{x}}, \mathsf{Hdecom}_{\boldsymbol{x}})$ opens to $\boldsymbol{x}$. If not, sends abort to both parties.
2. Samples $\mathsf{K}[\boldsymbol{x}] \leftarrow \mathbb{F}^m$, $\Delta \leftarrow \mathbb{F}$. If P_B is malicious, receives $(\mathsf{K}[\boldsymbol{x}], \Delta) \in \mathbb{F}^{m+1}$ from adversary. Computes $\mathsf{M}[\boldsymbol{x}] := \mathsf{K}[\boldsymbol{x}] + \Delta \cdot \boldsymbol{x} \in \mathbb{F}^m$. If P_A is malicious, receives $\mathsf{M}[\boldsymbol{x}] \in \mathbb{F}^m$ from the adversary, and recomputes $\mathsf{K}[\boldsymbol{x}] := \mathsf{M}[\boldsymbol{x}] - \Delta \cdot \boldsymbol{x} \in \mathbb{F}^m$. Sends $\mathsf{M}[\boldsymbol{x}]$ to P_A and $(\Delta, \mathsf{K}[\boldsymbol{x}])$ to P_B.

Figure 1: The commited-VOLE functionality.

1. **Computational hiding:** with overwhelming probability over the choice of k, for every $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{F}^m$ and $\{\mathsf{com}_{\boldsymbol{x}}, \mathsf{decom}_{\boldsymbol{x}}\} \leftarrow \mathsf{Com}(\boldsymbol{x})$, $\{\mathsf{com}'_{\boldsymbol{x}}, \mathsf{decom}'_{\boldsymbol{x}}\} \leftarrow \mathsf{Com}(\boldsymbol{x}')$, the distributions of $\mathsf{com}_{\boldsymbol{x}}$ and $\mathsf{com}'_{\boldsymbol{x}}$ are computationally indistinguishable.
2. **Statistical binding:** with overwhelming probability over the choice of k, the following equation holds with probability $\mathsf{negl}(\lambda)$,

$$\mathsf{Open}(\mathsf{com}_{\boldsymbol{x}}, \mathsf{decom}_{\boldsymbol{x}}, \boldsymbol{x}) \wedge \mathsf{Open}(\mathsf{com}_{\boldsymbol{x}}, \mathsf{decom}'_{\boldsymbol{x}}, \boldsymbol{x}') = 1$$

3.4 Ideal Functionalities

Committed-vector Oblivious Linear Evaluation. Committed-vector oblivious linear evaluation (C-VOLE) is a variant of VOLE that allows two parties to generate VOLE relations for a pre-committed vector $\boldsymbol{x} \in \mathbb{F}^m$. The commitment of $\boldsymbol{x}$, denoted as $\mathsf{Hcom}_{\boldsymbol{x}}$, is public. The holder of $\boldsymbol{x}$ serves as P_A, while any party knows the public commitment $\mathsf{Hcom}_{\boldsymbol{x}}$ can act as P_B and execute the functionality depicted in Figure 1 with P_A. The functionality $\mathcal{F}_{\mathsf{C-VOLE}}$ first checks whether P_A holds the correct $(\boldsymbol{x}, \mathsf{Hdecom}_{\boldsymbol{x}})$ that satisfies the public commitment $\mathsf{Hcom}_{\boldsymbol{x}}$. It then samples a vector $\mathsf{K}[\boldsymbol{x}] \leftarrow \mathbb{F}^m$ and a field element $\Delta \leftarrow \mathbb{F}$ to P_B and delivers $\mathsf{M}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta \in \mathbb{F}^m$ to P_A, as the VOLE functionality typically does.

Vector Oblivious Linear Evaluation. Vector Oblivious Linear Evaluation (VOLE) is generating VOLE relation of a designated vector rather than a random vector. The ideal functionality is shown in Figure 2. The functionality samples field element $\Delta \leftarrow \mathbb{F}$ to P_B for initialization. Then, for P_A's input vector $\boldsymbol{x} \in \mathbb{F}^m$, the functionality samples $\mathsf{K}[\boldsymbol{x}] \leftarrow \mathbb{F}^m$ to P_B and delivers $\mathsf{M}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta \in \mathbb{F}^m$ to P_A. VOLE is typically instantiated from Random VOLE (RVOLE). It also serves as a building block in the instantiation of $\mathcal{F}_{\mathsf{C-VOLE}}$.

Multi-point Vector Oblivious Linear Evaluation. Multi-point Vector Oblivious Linear Evaluation (MPVOLE) is a variant of VOLE that generates the VOLE relation of a multi-point vector, a regular sparse vector formed by concatenating multiple single-point (unary) vectors. Its ideal functionality is shown in Figure 3 and consists of two phases: the functionality samples field element $\Delta \leftarrow \mathbb{F}$ to P_B for initialization; Then, for P_A's input multi-point vector $\boldsymbol{e}_u \in \mathbb{F}^{t \cdot 2^h}$, where $t \cdot 2^h = n$, the functionality samples $\mathsf{K}[\boldsymbol{e}_u] \leftarrow \mathbb{F}^n$ to P_B and delivers $\mathsf{M}[\boldsymbol{e}_u] = \mathsf{K}[\boldsymbol{e}_u] + \boldsymbol{e}_u \cdot \Delta \in \mathbb{F}^n$ to P_A. We provide the construction of MPVOLE in Section A.5, which is a slightly variant of [BCGI18, WYKW21]. It serves as an important building block in instantiating $\mathcal{F}_{\mathsf{C-VOLE}}$.

Functionality $\mathcal{F}_{\mathsf{VOLE}}$

Initialize: Upon receiving (init) from both $\mathsf{P_A}$ and $\mathsf{P_B}$, sample $\Delta \leftarrow \mathbb{F}$ and send it to $\mathsf{P_B}$. The functionality stores Δ. If $\mathsf{P_B}$ is malicious, receive $\Delta \in \mathbb{F}$ from $\mathsf{P_B}$.

Extend: Upon receiving $(\mathsf{extend}, \boldsymbol{x} \in \mathbb{F}^m, m)$ from $\mathsf{P_A}$ and (extend, m) from $\mathsf{P_B}$:

1. Samples $\mathsf{K}[\boldsymbol{x}] \leftarrow \mathbb{F}^m$. If $\mathsf{P_B}$ is malicious, receive $\mathsf{K}[\boldsymbol{x}] \in \mathbb{F}^m$ from adversary.
2. Computes $\mathsf{M}[\boldsymbol{x}] := \mathsf{K}[\boldsymbol{x}] + \Delta \cdot \boldsymbol{x} \in \mathbb{F}^m$. If $\mathsf{P_A}$ is malicious, receives $\mathsf{M}[\boldsymbol{x}] \in \mathbb{F}^m$ from the adversary, and recomputes $\mathsf{K}[\boldsymbol{x}] := \mathsf{M}[\boldsymbol{x}] - \Delta \cdot \boldsymbol{x} \in \mathbb{F}^m$
3. Sends $\mathsf{M}[\boldsymbol{x}]$ to $\mathsf{P_A}$ and $\mathsf{K}[\boldsymbol{x}]$ to $\mathsf{P_B}$.

Figure 2: The VOLE functionality.

Functionality $\mathcal{F}_{\mathsf{MPVOLE}}$

Initialize: Upon receiving (init) from both $\mathsf{P_A}$ and $\mathsf{P_B}$, sample $\Delta \leftarrow \mathbb{F}$ to $\mathsf{P_B}$. The functionality stores Δ. If $\mathsf{P_B}$ is malicious, receive $\Delta \in \mathbb{F}$ from $\mathsf{P_B}$.

Extend: Upon receiving $(\mathsf{extend}, \boldsymbol{e}_u \in \mathbb{F}^{t \cdot 2^h}, h, t)$ from $\mathsf{P_A}$ where $\boldsymbol{e}_u$ is a regular sparse vector with t non-zero elements and $n = t \cdot 2^h$, and (extend, h, t) from $\mathsf{P_B}$.

1. Samples $\mathsf{K}[\boldsymbol{e}_u] \leftarrow \mathbb{F}^n$. If $\mathsf{P_B}$ is malicious, receives $\mathsf{K}[\boldsymbol{e}_u] \in \mathbb{F}^n$ from adversary.
2. Computes $\mathsf{M}[\boldsymbol{e}_u] := \mathsf{K}[\boldsymbol{e}_u] + \Delta \cdot \boldsymbol{e}_u \in \mathbb{F}^n$. If $\mathsf{P_A}$ is malicious, receives $\mathsf{M}[\boldsymbol{e}_u] \in \mathbb{F}^n$ from the adversary, and recomputes $\mathsf{K}[\boldsymbol{e}_u] := \mathsf{M}[\boldsymbol{e}_u] - \Delta \cdot \boldsymbol{e}_u \in \mathbb{F}^n$.
3. Sends $\mathsf{M}[\boldsymbol{e}_u]$ to $\mathsf{P_A}$ and $\mathsf{K}[\boldsymbol{e}_u]$ to $\mathsf{P_B}$.

Selective failure: In the malicious setting, if $\mathsf{P_B}$ is corrupted, the adversary is allowed to make the following selective failure query for each extend call:

1. Wait for the adversary to input t sets $I_1, \cdots, I_t \subseteq [0, n)$. Let $\{\alpha_i\} \in [0, n)$ be the index of the non-zero entry of ith single-point block $\boldsymbol{e}_u^i$.
2. For all $\alpha_i \in I_i$ for $i \in [t]$, send success to $\mathsf{P_B}$ and continue. Otherwise, send abort to both parties and abort.

Figure 3: The multi-point VOLE functionality.

Crowd Private Set Intersection. Private set intersection (PSI) allows two distrusted parties to jointly compute the intersection of their private inputs without revealing any additional information about those inputs except for the set sizes. Crowd PSI is an extension of PSI involving a server and multiple clients, where the server executes PSI with each individual client. In Figure 4, we describe the ideal functionality of crowd PSI, with the server (party P_1) acting as the receiver of the intersection and the clients (party $P_2, \cdots, P_n$) functioning as senders. The server sends its private set to the functionality for initialization. The functionality waiting for any client's private set, stores the server's set, and returns the intersection to the server.

Functionality $\mathcal{F}_{\mathsf{PSI}}$

There are n parties: $P_1, \cdots, P_n$

1. Upon receiving (init, X) from party P_1, and $(\mathsf{intersect}, Y_2)$ from a party P_2, where $X, Y_2 \subset U$, the functionality stores X and outputs $(X \cap Y_2, \mathsf{setid})$ to P_1. The functionality publishes setid.
2. Upon receiving $(\mathsf{init}, \mathsf{setid})$ from party P_1, and $(\mathsf{intersect}, Y_j, \mathsf{setid})$ from a party P_j, where $Y_j \subset U$, $j \in [3, n]$, the functionality outputs $X \cap Y_j$ to P_1.

Figure 4: The crowd private set intersection functionality.

4 Commitment from LPN

We introduce the commitment scheme we use, denoted as a hybrid commitment, in the committed-VOLE construction. As outlined in Section 2, the hybrid commitment scheme relies on the LPN assumption and consists of two parts: an LPN commitment to an intermediate regular sparse vector and a correction to the designated committed vector.

We construct the LPN commitment specifically for a regular sparse vector. In Theorem 1, we prove that with sparse message distribution, the new LPN commitment schemes still satisfy the requirements of binding and hiding. Then, we add a correction vector to the LPN commitment to construct a hybrid commitment scheme for a designated vector, specifically designed for the C-VOLE construction. The proposed hybrid commitment satisfy the binding and hiding property as detailed in Theorem 2.

4.1 LPN Commitment

Here, we define a commitment scheme $\mathsf{Com}(\cdot)$ based on the primal LPN assumption for committing to a regular-w vector $\boldsymbol{x}$. This vector $\boldsymbol{x} \in \mathbb{F}^m$, for parameters w, L and $m = w \cdot 2^L$, consists of w blocks, each of size 2^L, with each block contains exactly one non-zero element.

- **ParamGen.** The public commitment parameter is represented by $\mathsf{K} = \{\mathbf{A}_1, \mathbf{A}_2\}$, where $\mathbf{A}_1 \leftarrow \mathbf{C}(N, m, \mathbb{F})$ and $\mathbf{A}_2 \leftarrow \mathbb{F}^{k \times N}$, with $N = t \cdot 2^{L'}$. The algorithm $\mathbf{C}$ is a probabilistic code generation algorithm that outputs a matrix $\mathbf{A}_1 \in \mathbb{F}^{m \times N}$.
- **Commit.** The commitment to a regular-w message $\boldsymbol{x} \in \mathbb{F}^m$ is expressed as $\mathsf{com}_{\boldsymbol{x}} = \boldsymbol{x} \cdot \mathbf{A}_1 + \boldsymbol{r} \cdot \mathbf{A}_2 + \boldsymbol{e}$, where $\boldsymbol{r} \leftarrow \mathbb{F}^k$ and $\boldsymbol{e}$ is a weight-t regular noise vector of length N. The opening of the commitment is given by $\mathsf{decom}_{\boldsymbol{x}} = \boldsymbol{r}$.
- **Open.** Given a commitment $\mathsf{com}_{\boldsymbol{x}}$, a regular-$w$ message $\boldsymbol{x}$, and randomness $\mathsf{decom}_{\boldsymbol{x}}$, a verifier accepts if $\mathsf{com}_{\boldsymbol{x}} - \boldsymbol{x} \cdot \mathbf{A}_1 - \mathsf{decom}_{\boldsymbol{x}} \cdot \mathbf{A}_2$ is weight-t regular.

The corresponding dual LPN commitment scheme $\mathsf{Com}(\cdot)$ for a regular-w message $\boldsymbol{x} \in \mathbb{F}^m$ is described below:

- **ParamGen.** The public commitment parameter is represented by $\mathsf{K} \stackrel{\text{def}}{=} \{\mathbf{H}_1, \mathbf{H}_2\}$, where $\mathbf{H}_1 \leftarrow \mathbf{C}^{\perp}(n, m, \mathbb{F})$ and $\mathbf{H}_2 \leftarrow \mathbb{F}^{N \times n}$, with $N = t \cdot 2^{L'}$. The algorithm $\mathbf{C}^{\perp}$ is a probabilistic code generation algorithm that outputs a matrix $\mathbf{H}_1 \in \mathbb{F}^{m \times n}$.
- **Commit.** The commitment to a regular-w message $\boldsymbol{x} \in \mathbb{F}^m$ is expressed as $\mathsf{com}_{\boldsymbol{x}} = \boldsymbol{x} \cdot \mathbf{H}_1 + \boldsymbol{e} \cdot \mathbf{H}_2$, where $\boldsymbol{e}$ is a weight-t regular noise vector of length N. The opening of the commitment is given by $\mathsf{decom}_x = \boldsymbol{e}$.

- **Open.** Given a commitment $\mathsf{com}_{\boldsymbol{x}}$, a regular-$w$ message $\boldsymbol{x}$, and randomness $\mathsf{decom}_{\boldsymbol{x}}$, a verifier accepts if $\mathsf{decom}_{\boldsymbol{x}}$ is weight-t regular and $\mathsf{com}_{\boldsymbol{x}} = \boldsymbol{x} \cdot \mathbf{H}_1 + \mathsf{decom}_{\boldsymbol{x}} \cdot \mathbf{H}_2$.

Theorem 1. *Assume $L', k, t \in \mathbb{N}$ such that the decisional primal-LPN(N, k, t) problem (with secrets of length k and N samples) and the decisional dual-LPN$(N, n = N - k, t)$ problem are computationally hard, where $N = t \cdot 2^{L'}$. Then the above two commitment schemes are ε-statistically binding and computationally hiding, where $\lambda = 2^{2wL+2tL'}/|\mathbb{F}|^{t\cdot 2^{L'}-k-2w-2t}$.*

Proof. The security properties are proved as follows:

1. **Statistically binding.**

   - **For the commitment scheme based on the primal LPN problem.** Suppose there exists a commitment $\boldsymbol{c}$ corresponding to two messages, $\boldsymbol{x}_1 \in \mathbb{F}^m$ and $\boldsymbol{x}_2 \in \mathbb{F}^m$, with randomness pairs $(\boldsymbol{r}_1, \boldsymbol{e}_1)$ and $(\boldsymbol{r}_1, \boldsymbol{e}_2)$. Define $\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{x}_1 - \boldsymbol{x}_2 \in \mathbb{F}^m$, $\boldsymbol{r} \stackrel{\text{def}}{=} \boldsymbol{r}_1 - \boldsymbol{r}_2 \in \mathbb{F}^k$ and $\boldsymbol{e} \stackrel{\text{def}}{=} \boldsymbol{e}_1 - \boldsymbol{e}_2 \in \mathbb{F}^N$. Given that each $\boldsymbol{x}_i$ is a weight-w regular vector of length $m = w \cdot 2^L$ and each $\boldsymbol{e}_i$ is a weight-t regular vector of length $N = t \cdot 2^{L'}$, the probability that $\boldsymbol{x} \cdot \mathbf{A}_1 + \boldsymbol{r} \cdot \mathbf{A}_2 + \boldsymbol{e} = 0$ is at most $2^{2wL+2tL'}/|\mathbb{F}|^{t\cdot 2^{L'}-2w-2t}$. A union bound over all possible $\boldsymbol{r} \in \mathbb{F}^k$ yields the claimed bound $2^{2wL+2tL'}/|\mathbb{F}|^{t\cdot 2^{L'}-k-2w-2t}$.
   - **For the commitment scheme based on the dual LPN problem.** Considering the commitment $\boldsymbol{c}$ associated with two messages, $\boldsymbol{x}_1 \in \mathbb{F}^m$ and $\boldsymbol{x}_2 \in \mathbb{F}^m$, along with their respective randomness $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$. Define $\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{x}_1 - \boldsymbol{x}_2 \in \mathbb{F}^m$ and $\boldsymbol{e} \stackrel{\text{def}}{=} \boldsymbol{e}_1 - \boldsymbol{e}_2 \in \mathbb{F}^N$. Given that each $\boldsymbol{x}_i$ is a weight-w regular vector of length $m = w \cdot 2^L$ and each $\boldsymbol{e}_i$ is a weight-t regular vector of length $N = t \cdot 2^{L'}$, the probability that $\boldsymbol{x} \cdot \mathbf{H}_1 + \boldsymbol{e} \cdot \mathbf{H}_2 = 0$ is at most $2^{2wL+2tL'}/|\mathbb{F}|^{t\cdot 2^{L'}-k-2w-2t}$.

2. **Computationally hiding.** In the commitment scheme based on the primal LPN problem, we have $\boldsymbol{c} = \boldsymbol{x} \cdot \mathbf{A}_1 + \boldsymbol{r} \cdot \mathbf{A}_2 + \boldsymbol{e}$. In the commitment scheme based on the dual LPN problem, the commitment is $\boldsymbol{c} = \boldsymbol{x} \cdot \mathbf{H}_1 + \boldsymbol{e} \cdot \mathbf{H}_2$. Leveraging the security properties of the primal LPN and dual LPN problems, the terms $\boldsymbol{r} \cdot \mathbf{A}_2 + \boldsymbol{e}$ and $\boldsymbol{e} \cdot \mathbf{H}_2$ are pseudorandom, ensuring the pseudorandomness of the respective commitments.

□

4.2 Hybrid Commitment

We present the hybrid commitment scheme $\mathsf{HCom}(\cdot)$, designed for C-VOLE construction for messages $\boldsymbol{x} \in \mathbb{F}^m$, with the following components:

- **ParamGen.** The public commitment parameter consists of the public commitment parameters of Com and a matrix $\mathbf{H}^{\mathsf{T}} \leftarrow \mathbf{C}^{\perp}(N, m, \mathbb{F})$, where $N = w \cdot 2^L$. The algorithm $\mathbf{C}^{\perp}$ is a probabilistic code generation algorithm that outputs a matrix $\mathbf{H} \in \mathbb{F}^{N \times m}$.

- **Commit.** The hybrid commitment to a message $\boldsymbol{x} \in \mathbb{F}^m$ is expressed as $\mathsf{Hcom}_{\boldsymbol{x}} = \{\mathsf{com}_{\boldsymbol{e}}, \boldsymbol{c} = \boldsymbol{e} \cdot \mathbf{H} - \boldsymbol{x}\}$, where $\boldsymbol{e}$ is regular-w vector of length N and $(\mathsf{com}_{\boldsymbol{e}}, \mathsf{decom}_{\boldsymbol{e}}) \leftarrow \mathsf{Com}(\boldsymbol{e})$. The opening of the commitment is given by $\mathsf{Hdecom}_{\boldsymbol{x}} = \{\boldsymbol{e}, \mathsf{decom}_{\boldsymbol{e}}\}$.

- **Open.** Given a commitment $\mathsf{Hcom}_{\boldsymbol{x}} = \{\mathsf{com}_{\boldsymbol{e}}, \boldsymbol{c}\}$, a message $\boldsymbol{x}$, and randomness $\mathsf{Hdecom}_{\boldsymbol{x}} = \{\boldsymbol{e}, \mathsf{decom}_{\boldsymbol{e}}\}$, a verifier accepts if $\{\boldsymbol{e}, \mathsf{com}_{\boldsymbol{e}}, \mathsf{decom}_{\boldsymbol{e}}\}$ passes verification of Com, ensuring $\boldsymbol{e}$ is a weight-w regular vector; and $\boldsymbol{c} = \boldsymbol{e} \cdot \mathbf{H} - \boldsymbol{x}$.

Theorem 2. *Assume $L, m, t \in \mathbb{N}$ such that the decisional dual-LPN(N, m, w) problem, with matrix $\mathbf{H}^{\mathsf{T}} \leftarrow \mathbf{C}^{\perp}(N, m, \mathbb{F})$, is computationally hard, where $N = w \cdot 2^L$. If the commitment scheme Com is λ-statistically binding and computationally hiding, then the aforementioned hybrid commitment scheme is also λ-statistically binding and computationally hiding.*

Proof. The required security properties are as follows:

1. **Statistically binding.** $\boldsymbol{e}$ is λ-statistically binding according to the commitment scheme Com. Given that $\boldsymbol{x} = \boldsymbol{e} \cdot \mathbf{H} - \boldsymbol{c}$, it follows that $\boldsymbol{x}$ is also λ-statistically binding.

2. **Computationally hiding.** The hybrid commitment is based on the dual LPN problem, constructing the commitment $\boldsymbol{c} = \boldsymbol{e} \cdot \mathbf{H} - \boldsymbol{x}$. Leveraging the security properties of the dual LPN problem, the term $\boldsymbol{e} \cdot \mathbf{H}$ is pseudorandom, ensuring the pseudorandomness of the resulting commitment.

□

5 Committed-VOLE

We construct the committed-VOLE protocol using the hybrid commitment we introduced in Section 4.2. The C-VOLE protocol starts from one party P_A, generating a hybrid commitment of its chosen vector $\boldsymbol{x}$ and publishing the commitment to every other party. Then, P_A generates VOLE correlation with any other party P_B of the $\boldsymbol{x}$ consistent with the published hybrid commitment. In the construction of the hybrid commitment, there are two methods to commit the intermediate vector $\boldsymbol{e}_u$: dual-LPN commitment and primal-LPN commitment. This leads to two ways to instantiate the protocol: dual-LPN based C-VOLE and primal-LPN based C-VOLE. Here, we use the dual-LPN based hybrid commitment to P_A's input vector $\boldsymbol{x}$ and provide its corresponding C-VOLE instantiation in Figure 5. The primal-LPN based C-VOLE is introduced and evaluated in Section A.2. The concrete performance is provided in performance evaluation in Section 7.

The C-VOLE protocol based on dual-LPN hybrid commitment is constructed as follows: First, $\mathsf{P_A}$ commits to its chosen input vector $\boldsymbol{x}$. $\mathsf{P_A}$ samples an intermediate regular sparse vector $\boldsymbol{e}_u$ and computes a dual-LPN commitment $\mathsf{com}_{\boldsymbol{e}_u}$. $\mathsf{P_A}$ computes the hybrid commitment $\mathsf{Hcom}_{\boldsymbol{x}}$ of $\boldsymbol{x}$ as $\mathsf{Hcom}_{\boldsymbol{x}} = \{\mathsf{com}_{\boldsymbol{e}_u}, \boldsymbol{x}^* = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x}\}, \mathsf{Hdecom}_{\boldsymbol{x}} = \{\boldsymbol{e}_u, \mathsf{decom}_{\boldsymbol{e}_u}\}$, where $\mathbf{H}$ is a public matrix and $\boldsymbol{x}^*$ is a correction vector from the intermediate $\boldsymbol{e}_u$ to the designated $\boldsymbol{x}$. Then, to generate the VOLE correlation of the designated $\boldsymbol{x}$ committed by $\mathsf{Hcom}_{\boldsymbol{x}}$, we let both parties first generate the VOLE correlation of the intermediate $\boldsymbol{e}_u$ committed by $\mathsf{com}_{\boldsymbol{e}_u}$ (part of $\mathsf{Hcom}_{\boldsymbol{x}}$). Given $[\![\boldsymbol{e}_u]\!]$ is consistent with $\mathsf{com}_{\boldsymbol{e}_u}$, both parties can locally recover $[\![\boldsymbol{x}]\!]$, which is consistent with $\mathsf{Hcom}_{\boldsymbol{x}}$, using the publicly known correction vector $\boldsymbol{x}^*$ (part of $\mathsf{Hcom}_{\boldsymbol{x}}$) as follows:

Given $\mathsf{M}[\boldsymbol{e}_u] = \mathsf{K}[\boldsymbol{e}_u] + \boldsymbol{e}_u \cdot \Delta$, where $(\boldsymbol{e}_u, \mathsf{M}[\boldsymbol{e}_u])$ is held by $\mathsf{P_A}$ and $(\mathsf{K}[\boldsymbol{e}_u], \Delta)$ is held by $\mathsf{P_B}$, and the public correction vector $\boldsymbol{x}^* = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x}$, both parties compute $\mathsf{M}[\boldsymbol{x}], \mathsf{K}[\boldsymbol{x}]$, respectively, as:

$$\mathsf{M}[\boldsymbol{x}] = \mathsf{M}[\boldsymbol{e}_u] \cdot \mathbf{H}$$
$$\mathsf{K}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H} + \boldsymbol{x}^* \cdot \Delta,$$

such that $\mathsf{M}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta$.

In this way, generating the VOLE correlation for the committed $\boldsymbol{x}$ is reduced to generating the VOLE correlation for the committed intermediate vector $\boldsymbol{e}_u$. Notice that $\boldsymbol{e}_u$ is a regular-w vector with length N, where $N = w \cdot 2^L$, defined in Section 4.2. Both parties call the multi-point VOLE functionality $\mathcal{F}_{\mathsf{MPVOLE}}$, as depicted in Figure 3, to generate $[\![\boldsymbol{e}_u]\!]$. The instantiation of $\mathcal{F}_{\mathsf{MPVOLE}}$

follows the single-point subfield VOLE Π_{spsVOLE} in [WYKW21] with a slight variation, which is detailed in Section A.5. In the next step, $\mathsf{P_A}$ needs to prove to $\mathsf{P_B}$ that with $\boldsymbol{e}_u$ input to $\mathcal{F}_{\mathsf{MPVOLE}}$, the resulted $[\![\boldsymbol{e}_u]\!]$ is consistent with commitment $\mathsf{com}_{\boldsymbol{e}_u}$. We construct the proof efficiently leveraging the linear property of the dual-LPN commitment.

Recall that for dual-LPN commitment of $\boldsymbol{e}_u$, $\mathsf{com}_{\boldsymbol{e}_u} = \boldsymbol{e}_u \cdot \mathbf{H}_1 + \boldsymbol{e}_r \cdot \mathbf{H}_2$ and $\mathsf{decom}_{e_u} = \boldsymbol{e}_r$. We let both parties call $\mathcal{F}_{\mathsf{MPVOLE}}$ again with $\mathsf{P_A}$ input $\mathsf{decom}_{e_u}(\boldsymbol{e}_r)$ to generate $[\![\boldsymbol{e}_r]\!]$. Given $\mathsf{M}[\boldsymbol{e}_u] = \mathsf{K}[\boldsymbol{e}_u] + \boldsymbol{e}_u \cdot \Delta$, where $(\boldsymbol{e}_u, \mathsf{M}[\boldsymbol{e}_u])$ is held by $\mathsf{P_A}$ and $(\mathsf{K}[\boldsymbol{e}_u], \Delta)$ is held by $\mathsf{P_B}$, and $\mathsf{M}[\boldsymbol{e}_r] = \mathsf{K}[\boldsymbol{e}_r] + \boldsymbol{e}_r \cdot \Delta$, where $(\boldsymbol{e}_r, \mathsf{M}[\boldsymbol{e}_r])$ is held by $\mathsf{P_A}$ and $(\mathsf{K}[\boldsymbol{e}_r])$ is held by $\mathsf{P_B}$, both parties compute $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]$ and $\mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}]$, respectively:

$$\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{M}[\boldsymbol{e}_u] \cdot \mathbf{H}_1 + \mathsf{M}[\boldsymbol{e}_r] \cdot \mathbf{H}_2$$
$$\mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H}_1 + \mathsf{K}[\boldsymbol{e}_r] \cdot \mathbf{H}_2$$

Since the LPN-commitment $\mathsf{com}_{\boldsymbol{e}_u}$ is published in the hybrid commitment $\mathsf{Hcom}_{\boldsymbol{x}}$ of $\boldsymbol{x}$, the following equation ($[\![\mathsf{com}_{\boldsymbol{e}_u}]\!]$) holds,

$$\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}] + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta$$

if the $[\![\boldsymbol{e}_u]\!]$ and $[\![\boldsymbol{e}_r]\!]$ used in computing $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]$ and $\mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}]$ are consistent with the $\boldsymbol{e}_u$ and $\boldsymbol{e}_r$ in computing $\mathsf{com}_{\boldsymbol{e}_u}$. We let $\mathsf{P_A}$ send $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])$ to $\mathsf{P_B}$. $\mathsf{P_B}$ verifies whether $[\![\mathsf{com}_{\boldsymbol{e}_u}]\!]$ holds, thereby ensuring that a consistent intermediate VOLE correlation $[\![\boldsymbol{e}_u]\!]$ is generated with the intermediate commitment $\mathsf{com}_{\boldsymbol{e}_u}$.

The detailed protocol for instantiating $\mathcal{F}_{\mathsf{C-VOLE}}$ with a dual-LPN commitment is shown in Figure 5. We also proved that the dual-LPN commitment based protocol $\Pi_{\mathsf{C-VOLE-dual}}$ is secure against any malicious adversary, as established in Theorem 3 with a complete proof.

5.1 Proof of Security

Theorem 3. *Protocol $\Pi_{\mathsf{C-VOLE-dual}}$ in Figure 5 securely instantiates $\mathcal{F}_{\mathsf{C-VOLE}}$ in Figure 1 in $\mathcal{F}_{\mathsf{MPVOLE}}$-hybrid model against any malicious adversaries.*

Proof. Let $\mathcal{A}$ be a PPT adversary that corrupt $\mathsf{P_A}$ or $\mathsf{P_B}$. We construct a PPT simulator $\mathcal{S}$ with access to $\mathcal{F}_{\mathsf{C-VOLE}}$ and simulates the adversary's view. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the output of $\mathcal{S}$ and the honest party in the ideal world.

Corrupted $\mathsf{P_A}$. Let $\mathcal{S}$ access $\mathcal{F}_{\mathsf{C-VOLE}}$ as an honest $\mathsf{P_A}$ and interact with $\mathcal{A}$ as an honest $\mathsf{P_B}$. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

1. $\mathcal{S}$ holds public commitment of $\boldsymbol{x}$, that $\mathsf{Hcom}_{\boldsymbol{x}} = (\mathsf{com}_{\boldsymbol{e}_u}, \boldsymbol{x}^* = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x})$. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives (init) from $\mathcal{A}$.

2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives $(\mathsf{extend}, \boldsymbol{e}'_u \in \mathbb{F}^{w \cdot 2^L}, L, w)$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{M}[\boldsymbol{e}'_u] \in \mathbb{F}^{w \cdot 2^L}$ from $\mathcal{A}$. $\mathcal{S}$ computes $\boldsymbol{x}' = \boldsymbol{e}'_u \cdot \mathbf{H} - \boldsymbol{x}^*$.

3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives $(\mathsf{extend}, \boldsymbol{e}'_r \in \mathbb{F}^{t \cdot 2^{L'}}, L', t)$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{M}[\boldsymbol{e}'_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ from $\mathcal{A}$. $\mathcal{S}$ sends $(\boldsymbol{x}', \mathsf{decom}'_x = (\boldsymbol{e}'_u, \boldsymbol{e}'_r))$ to $\mathcal{F}_{\mathsf{C-VOLE}}$. If $\mathcal{S}$ receives abort from $\mathcal{F}_{\mathsf{C-VOLE}}$, $\mathcal{S}$ aborts at step (4).

4. $\mathcal{S}$ receives $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])'$ from $\mathcal{A}$. $\mathcal{S}$ checks if $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])' = H_3(\mathsf{M}[\boldsymbol{e}'_u] \cdot \mathbf{H}_1 + \mathsf{M}[\boldsymbol{e}'_r] \cdot \mathbf{H}_2)$. If not, $\mathcal{S}$ aborts.

5. $\mathcal{S}$ computes $\mathsf{M}[\boldsymbol{x}] = \mathsf{M}[\boldsymbol{e}'_u] \cdot \mathbf{H}$ and sends $\mathsf{M}[\boldsymbol{x}]$ to $\mathcal{F}_{\mathsf{C-VOLE}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

Protocol $\Pi_{\mathsf{C-VOLE-dual}}$

Parameters: A public hybrid commitment of $\boldsymbol{x} \in \mathbb{F}^m$ defined in Section 4.2, that $\mathsf{Hcom}_{\boldsymbol{x}} = (\mathsf{com}_{\boldsymbol{e}_u} \in \mathbb{F}^n, \boldsymbol{x}^* = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x} \in \mathbb{F}^m)$. A dual LPN commitment scheme defined in Section 4.1 that $(\mathsf{com}_{\boldsymbol{e}_u} = \boldsymbol{e}_u \cdot \mathbf{H}_1 + \boldsymbol{e}_r \cdot \mathbf{H}_2, \mathsf{decom}_{\boldsymbol{e}_u} = \boldsymbol{e}_r) \leftarrow \mathsf{Com}(\boldsymbol{e}_u)$, where $\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}, \mathsf{com}_{\boldsymbol{e}_u} \in \mathbb{F}^n, \mathsf{decom}_{\boldsymbol{e}_u} \in \mathbb{F}^{t \cdot 2^{L'}}$. A Collision-Resistant Hash Function H_3.

Inputs: $\mathsf{P_A}$ holds vector $\boldsymbol{x} \in \mathbb{F}^m$, $\mathsf{Hdecom}_{\boldsymbol{x}} = (\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}, \boldsymbol{e}_r \in \mathbb{F}^{t \cdot 2^{L'}})$.

Compute:

1. Both $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{MPVOLE}}$ with input (init). $\mathsf{P_B}$ receives $\Delta \in \mathbb{F}$ from $\mathcal{F}_{\mathsf{MPVOLE}}$.
2. $\mathsf{P_A}$ calls $\mathcal{F}_{\mathsf{MPVOLE}}$ with input $(\mathsf{extend}, \boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}, L, w)$. $\mathsf{P_B}$ calls $\mathcal{F}_{\mathsf{MPVOLE}}$ with input (extend, L, w). The functionality returns $\mathsf{M}[\boldsymbol{e}_u] \in \mathbb{F}^{w \cdot 2^L}$ to $\mathsf{P_A}$ and $\mathsf{K}[\boldsymbol{e}_u] \in \mathbb{F}^{w \cdot 2^L}$ to $\mathsf{P_B}$ such that $\mathsf{M}[\boldsymbol{e}_u] = \mathsf{K}[\boldsymbol{e}_u] + \Delta \cdot \boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}$. If either party receives abort from $\mathcal{F}_{\mathsf{MPVOLE}}$ in any of these executions, it aborts.
3. $\mathsf{P_A}$ calls $\mathcal{F}_{\mathsf{MPVOLE}}$ with input $(\mathsf{extend}, \boldsymbol{e}_r \in \mathbb{F}^{t \cdot 2^{L'}}, L', t)$. $\mathsf{P_B}$ calls $\mathcal{F}_{\mathsf{MPVOLE}}$ with input (extend, L', t). The functionality returns $\mathsf{M}[\boldsymbol{e}_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ to $\mathsf{P_A}$ and $\mathsf{K}[\boldsymbol{e}_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ to $\mathsf{P_B}$ such that $\mathsf{M}[\boldsymbol{e}_r] = \mathsf{K}[\boldsymbol{e}_r] + \Delta \cdot \boldsymbol{e}_r \in \mathbb{F}^{t \cdot 2^{L'}}$. If either party receives abort from $\mathcal{F}_{\mathsf{MPVOLE}}$ in any of these executions, it aborts.
4. $\mathsf{P_A}$ computes $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{M}[\boldsymbol{e}_u] \cdot \mathbf{H}_1 + \mathsf{M}[\boldsymbol{e}_r] \cdot \mathbf{H}_2 \in \mathbb{F}^n$ and sends $H(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])$ to $\mathsf{P_B}$. $\mathsf{P_B}$ computes $\mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H}_1 + \mathsf{K}[\boldsymbol{e}_r] \cdot \mathbf{H}_2 \in \mathbb{F}^n$ and $\mathbf{value} = \mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}] + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta \in \mathbb{F}^n$. $\mathsf{P_B}$ checks whether $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]) = H_3(\mathbf{value})$. If not, $\mathsf{P_B}$ aborts.
5. $\mathsf{P_A}$ outputs $(\boldsymbol{x}, \mathsf{M}[\boldsymbol{x}])$, where $\mathsf{M}[\boldsymbol{x}] = \mathsf{M}[\boldsymbol{e}_u] \cdot \mathbf{H} \in \mathbb{F}^m$. $\mathsf{P_B}$ outputs $(\Delta, \mathsf{K}[\boldsymbol{x}])$, where $\mathsf{K}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H} + \boldsymbol{x}^* \Delta \in \mathbb{F}^m$.

Figure 5: The committed-VOLE protocol based on dual-LPN-based hybrid commitment

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $\mathcal{F}_{\mathsf{MPVOLE}}$-hybrid model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$. Notice that $\mathcal{S}$ does not simulate any message to $\mathcal{A}$. This hybrid is identical to the previous one.

Hybrid $\mathcal{H}_2$ Same as Hybrid $\mathcal{H}_1$ except $\mathcal{S}$ aborts at step (4) if $\mathcal{S}$ receives abort from $\mathcal{F}_{\mathsf{C-VOLE}}$ or the received $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])' \neq H_3(\mathsf{M}[\boldsymbol{e}'_u] \cdot \mathbf{H}_1 + \mathsf{M}[\boldsymbol{e}'_r] \cdot \mathbf{H}_2)$.

In hybrid $\mathcal{H}_1$, an honest $\mathsf{P_B}$ aborts at step (4) if the received $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])' \neq H_3(\mathsf{K}[\boldsymbol{e}'_u] \cdot \mathbf{H}_1 + \mathsf{K}[\boldsymbol{e}'_r] \cdot \mathbf{H}_2 + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta)$. There are the following abort conditions with abort probability indistinguishable from hybrid $\mathcal{H}_2$:

1) $\mathcal{A}$ uses $\boldsymbol{e}'_u \neq \boldsymbol{e}_u$ and $\boldsymbol{e}'_r \neq \boldsymbol{e}_r$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{A}$ computes $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]'$ honestly from $\mathsf{M}[\boldsymbol{e}'_u] \cdot \mathbf{H}_1 + \mathsf{M}[\boldsymbol{e}'_r] \cdot \mathbf{H}_2$. Any $(\boldsymbol{e}'_u, \boldsymbol{e}'_r)$ inconsistent with $(\boldsymbol{e}_u, \boldsymbol{e}_r)$ will result the inequality above with the probability of statistical binding failure mentioned in Theorem 1. In this hybrid, any inconsistency will be checked by the $\mathcal{F}_{\mathsf{C-VOLE}}$ with the same binding failure and result in an abort at step (4), which is indistinguishable from hybrid $\mathcal{H}_1$. 2) $\mathcal{A}$ uses $\boldsymbol{e}'_u \neq \boldsymbol{e}_u$ and $\boldsymbol{e}'_r \neq \boldsymbol{e}_r$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{A}$ samples $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]'$ to $\mathcal{S}$. The consistency check passes if and only if $\mathcal{A}$ samples $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]' = \mathsf{K}[\boldsymbol{e}'_u] \cdot \mathbf{H}_1 + \mathsf{K}[\boldsymbol{e}'_r] \cdot \mathbf{H}_2 + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta$. However, since $\mathsf{K}[\boldsymbol{e}'_u]$, $\mathsf{K}[\boldsymbol{e}'_r]$, Δ are uniformly distributed over $\mathbb{F}^{w \cdot 2^L}, \mathbb{F}^{t \cdot 2^{L'}}$ and $\mathbb{F}$, respectively, the consistency check fails with all but negligible probability. In this hybrid, given inconsistent $(\boldsymbol{e}'_u, \boldsymbol{e}'_r)$ pair, $\mathcal{S}$ aborts except the failure probability of binding, which is also negligible and thus indistinguishable from hybrid $\mathcal{H}_1$. 3) $\mathcal{A}$ uses $\boldsymbol{e}'_u = \boldsymbol{e}_u$ and $\boldsymbol{e}'_r = \boldsymbol{e}_r$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{A}$ samples $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]'$ to $\mathcal{S}$. The consistency check passes if and only if $\mathcal{A}$ samples

$\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]' = \mathsf{K}[\boldsymbol{e}'_u] \cdot \mathbf{H}_1 + \mathsf{K}[\boldsymbol{e}'_r] \cdot \mathbf{H}_2 + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta$, which is equivalent to checking $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]' = \mathsf{M}[\boldsymbol{e}'_u] \cdot \mathbf{H}_1 + \mathsf{M}[\boldsymbol{e}'_r] \cdot \mathbf{H}_2$ in this hybrid. Thus, the abort probability is indistinguishable from hybrid $\mathcal{H}_1$.

Therefore, this hybrid is identically distributed as $\mathcal{H}_1$.

Corrupted P_B. Let $\mathcal{S}$ access $\mathcal{F}_{\mathsf{C-VOLE}}$ as an honest P_B and interact with $\mathcal{A}$ as an honest P_A. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ holds public commitment of $\boldsymbol{x}$, $\mathsf{Hcom}_{\boldsymbol{x}} = (\mathsf{com}_{\boldsymbol{e}_u}, \boldsymbol{x}^* = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x})$. $\mathcal{S}$ sends (init) to $\mathcal{F}_{\mathsf{C-VOLE}}$. $\mathcal{S}$ receives $\Delta \in \mathbb{F}$ from $\mathcal{A}$.

2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives (extend, L, w) from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{K}[\boldsymbol{e}_u] \in \mathbb{F}^{w \cdot 2^L}$ from $\mathcal{A}$.

3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives (extend, L', t) from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{K}[\boldsymbol{e}_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ from $\mathcal{A}$.

4. $\mathcal{S}$ computes $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]) = H_3(\mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H}_1 + \mathsf{K}[\boldsymbol{e}_r] \cdot \mathbf{H}_2 + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta)$ and sends it to $\mathcal{A}$. $\mathcal{S}$ aborts if $\mathcal{A}$ aborts.

5. $\mathcal{S}$ computes $\mathsf{K}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H} + \boldsymbol{x}^* \cdot \Delta \in \mathbb{F}^m$ and sends $(\Delta, \mathsf{K}[\boldsymbol{x}]) \in \mathbb{F}^{m+1}$ to $\mathcal{F}_{\mathsf{C-VOLE}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $\mathcal{F}_{\mathsf{MPVOLE}}$-hybrid model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and simulates the messages to $\mathcal{A}$.

$\mathcal{S}$ simulates $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]) = H_3(\mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H}_1 + \mathsf{K}[\boldsymbol{e}_r] \cdot \mathbf{H}_2 + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta)$ and sends it to $\mathcal{A}$. In the previous hybrid, an honest P_A computes $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{M}[\boldsymbol{e}_u] \cdot \mathbf{H}_1 + \mathsf{M}[\boldsymbol{e}_r] \cdot \mathbf{H}_2$, which holds the property that $\mathsf{M}[\boldsymbol{e}_u] = \mathsf{K}[\boldsymbol{e}_u] + \boldsymbol{e}_u \cdot \Delta$ and $\mathsf{M}[\boldsymbol{e}_r] = \mathsf{K}[\boldsymbol{e}_r] + \boldsymbol{e}_r \cdot \Delta$. Consequently, the $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]$ computed by simulator equals the one computed by honest P_A in real world. Thus, this hybrid is identical to the previous one.

This concludes the proof. □

5.2 Complexity Analysis

We give an analysis of the communication efficiency of dual-LPN based C-VOLE instantiation. As the protocol call $\mathcal{F}_{\mathsf{MPVOLE}}$ for generating $[\![\boldsymbol{e}_u]\!]$ and $[\![\boldsymbol{e}_r]\!]$, it requires MPVOLE instantiations of $\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}$ and $\boldsymbol{e}_r \in \mathbb{F}^{t \cdot 2^{L'}}$. The protocol also requires a $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])$ for consistency check and publishing a commitment of size $n + m$. According to the instantiation of MPVOLE protocol in Section A.5 , we have that for sparse vector of length $w \cdot 2^L$, it requires communication overhead of $wL\kappa$. Thus, the total communication overhead of dual-LPN based C-VOLE protocol is $(wL + tL')\kappa + (n + m) \log |\mathbb{F}|$.

6 Applications of C-VOLE

We present more details of three applications that can benefit from C-VOLE. Our design improves the overall performance of commit-and-prove VOLE-ZK, simplifies the communication pattern of VOLE-based generic MPC protocols, and facilitates crowd PSI. They are discussed in the following.

6.1 Commit-and-prove VOLE-ZK

VOLE-ZK refers to a category of private-coin interactive zero-knowledge proofs [WYKW21, YSWW21, DIO21, BMRS21]. One of its extensions is the commit-and-prove VOLE-ZK that proves a statement over publicly committed data [WYX+21]. To achieve this, [WYX+21] uses a hybrid commitment scheme that is based on the Merkle tree construction [Mer88]. Additionally, it proposes a proof of commitment opening by generically re-computing the whole Merkle tree in ZK.

For an arbitrary circuit $C(\cdot)$ and a witness $\boldsymbol{w}$, define a VOLE-ZK that proves the validity of $C(\boldsymbol{w}) = 1$ without revealing any information about $\boldsymbol{w}$. Additionally, define a Merkle tree-based commitment scheme $\mathsf{MerkleCom}(\cdot)$ that commits to arbitrary messages. Assume that a prover commits to the witness $\boldsymbol{w}$ by $c = \mathsf{MerkleCom}(\boldsymbol{w})$. Given (c, C), the commit-and-prove VOLE-ZK can be viewed as proving the possession of witness $\boldsymbol{w}$ such that $c = \mathsf{MerkleCom}(\boldsymbol{w}) \wedge C(\boldsymbol{w}) = 1$.

In [WYX+21], the proof of the first part requires a non-black-box access to the commitment scheme. Namely, the prover needs to prove the satisfiability of a composition of hash function circuits, which incurs $O(\ell C_{\mathsf{Hash}})$ communication overhead assuming hash function circuits of size C_{Hash} per block and $\boldsymbol{w} \in \mathbb{F}^{\ell}$. Our C-VOLE protocol reduces its cost to $O(\kappa \log \ell)$ by replacing the Merkle tree commitment with the LPN-based commitment $\mathsf{HCom}(\cdot)$, which comes with an efficient proof of opening that facilitates the proof of $c = \mathsf{HCom}(\boldsymbol{w}) \wedge C(\boldsymbol{w}) = 1$. We demonstrate the concrete performance improvement in Section 7.3.1.

6.2 Multi-Party VOLE in MPC and MVZK

It is common for multi-party protocols to rely on a multi-party VOLE protocol to generate correlated randomness. Examples include generic dishonest-majority MPC protocols [RS22, HSS20, WRK17] and multi-verifier zero-knowledge proofs [EPSW24]. In multi-party VOLE, each pair of parties (P_i, P_j) need to execute VOLE for the correlation $\mathsf{M}[\boldsymbol{x}^i]_j^i = \mathsf{K}[\boldsymbol{x}^i]_i^j + \boldsymbol{x}^i \cdot \Delta^j$. Achieving malicious security in such protocols requires that P_i uses the same $\boldsymbol{x}^i$ across all VOLE instances it executes with other parties. To ensure this, a costly consistency check is performed at the end of the multi-party VOLE. In prior works, this check involves n-party joint coin-tossing and multiple rounds of peer-to-peer communication and broadcasting, which are significantly constrained by network latency.

Our C-VOLE protocol eliminate this round-heavy consistency check. Namely, P_i first publishes its commitment to $\boldsymbol{x}^i$. Then, for each $j \in [n] \setminus \{i\})$, P_i interacts with P_j to execute committed VOLE, which ensures the VOLE correlations are consistent with the committed $\boldsymbol{x}^i$. As long as parties reach consensus on the commitment, there is no need for a final multi-round verification.

6.3 C-VOLE in Crowd PSI

Crowd PSI involves a server executing 2-party PSI with multiple clients in a distributed manner; any standalone malicious secure 2-party PSI is insufficient, as the server may use different private sets interacting with different clients. Introducing the C-VOLE could address this issue by building upon VOLE-based PSI works [RR22a, RS21], the most performant PSI works currently, and extending them to a C-VOLE based crowd PSI protocol. The protocol is constructed as follows to securely achieve $\mathcal{F}_{\mathsf{PSI}}$ in Figure 4. The protocol is shown in Figure 6 and its intuition is detailed as follows:

Initially, P_1 computes a vector $\boldsymbol{p}$ by solving a linear system of its private set elements, and publishes its commitment. Then, any client P_j executes the C-VOLE functionality with P_1 to get a VOLE relation associated with $\boldsymbol{p}$. Given the VOLE relation, expressed as $\mathsf{M}[\boldsymbol{p}]_j = \mathsf{K}[\boldsymbol{p}]_j + \boldsymbol{p} \cdot \Delta_j$, party P_j uses $\mathsf{K}[\boldsymbol{p}]_j$ and Δ_j to encode its private set elements and then sends the encodings to P_1. P_1 uses the corresponding $\mathsf{M}[\boldsymbol{p}]_j$ to encode its private set elements $x_i \in X, i \in [n]$. Then, P_1 is

Protocol Π_{PSI}

Parameters: Party P_1 with a set $X = (x_1, \ldots, x_n) \subset \mathbb{F}^n$. Party P_j with a set $Y_j \subset \mathbb{F}^{n'}$. A random function $\mathsf{row} : \mathbb{F} \times \{0,1\}^\kappa \to \{0,1\}^m$, where $m = O(n) + O(\lambda)$ as defined by OKVS scheme in [RR22a]. A Decode algorithm $\mathsf{Decode} : \mathbb{F}^m \times \mathbb{F} \times \{0,1\}^\kappa \to \mathbb{F}$, defined by [RR22a]. A hybrid commitment scheme defined in Section 4.2, such that $(\mathsf{Hcom}_{\boldsymbol{a}}, \mathsf{Hdecom}_{\boldsymbol{a}}) \leftarrow \mathsf{HCom}(\boldsymbol{a})$, $1 \leftarrow \mathsf{Open}(\boldsymbol{a}, \mathsf{Hdecom}_{\boldsymbol{a}}, \mathsf{Hcom}_{\boldsymbol{a}})$, where $\boldsymbol{a} \in \mathbb{F}^m$, $\mathsf{Hcom}_{\boldsymbol{a}} \in \mathbb{F}^N$, $\mathsf{Hdecom}_{\boldsymbol{a}} \in \mathbb{F}^{m'}$. Random oracle $H_1 : \mathbb{F} \to \mathbb{F}$, $H_2 : \mathbb{F} \times \mathbb{F} \to \{0,1\}^{out}$.

Initialize:

1. P_1 samples $r \leftarrow \{0,1\}^\kappa$ and solves the following systems to get $\boldsymbol{p} \in \mathbb{F}^m$.

$$\begin{bmatrix} \mathsf{row}(x_1, r) \\ \cdots \\ \mathsf{row}(x_n, r) \end{bmatrix} \boldsymbol{p}^T = (H_1(x_1), \cdots, H_1(x_n))^T$$

2. P_1 computes $(\mathsf{Hcom}_{\boldsymbol{p}}, \mathsf{Hdecom}_{\boldsymbol{p}}) \leftarrow \mathsf{HCom}(\boldsymbol{p})$ and publishes $\mathsf{Hcom}_{\boldsymbol{p}}$.

Extend: P_1 execute this phase with each party P_j, who possesses set Y_j consisting elements $(y_1, \cdots, y_{n'})$.

3. P_j samples $w^s \leftarrow \mathbb{F}$, sends $c^s := H_1(w^s)$ to P_1.
4. P_1 samples $w^r \leftarrow \mathbb{F}$, sends w^r, r to P_j.
5. P_1 calls $\mathcal{F}_{\mathsf{C-VOLE}}$ with input $(\mathsf{init}, \boldsymbol{p}, \mathsf{Hdecom}_{\boldsymbol{p}})$. P_j call $\mathcal{F}_{\mathsf{C-VOLE}}$ with input (init). The functionality sends $\mathsf{M}[\boldsymbol{p}] \in \mathbb{F}^m$ to P_1 and $(\Delta, \mathsf{K}[\boldsymbol{p}]) \in \mathbb{F}^{m+1}$ to P_j, such that $\mathsf{M}[\boldsymbol{p}] = \mathsf{K}[\boldsymbol{p}] + \boldsymbol{p} \cdot \Delta$.
6. P_j sends w^s to P_1. P_1 aborts if $c^s \neq H_1(w^s)$. Otherwise, both party computes $w = w^s + w^r$.
7. P_j computes $ey_i = F(y_i) = H_2(\mathsf{Decode}(\mathsf{K}[\boldsymbol{p}], y_i, r) + H_1(y_i) \cdot \Delta + w, y_i)$ for each $y_i \in Y_j$, inserts ey_i to set EY, and sends EY to P_1.
8. P_1 computes $ex_i = F(x_i) = H_2(\mathsf{Decode}(\mathsf{M}[\boldsymbol{p}], x_i, r) + w, x_i)$ for each $x_i \in X$, inserts ex_i to set EX. P_1 computes the intersection $Z : \{x_i | x_i \in X, ex_i \in EX \cap EY\}$

Figure 6: Secure crowd PSI protocol.

able to compute the intersection by locally comparing the encodings. By leveraging $\mathcal{F}_{\mathsf{C-VOLE}}$ and pre-published commitment of $\boldsymbol{p}$, this protocol ensures that P_1 consistently reuses $\boldsymbol{p}$ in the extend phase with different P_j.

The correctness of encoding and computing the intersection aligns with the VOLE-based PSI protocol [RS21]. Each set element x_i/y_i is mapped to a specific row vector through random function row. P_j encodes its set elements $y_i, i \in [n']$ by applying $\mathsf{K}[\boldsymbol{p}]$ to these row vectors to compute the inner product and then add $H(y_i) \cdot \Delta$. P_1 encodes its set elements $x_i, i \in [n]$ by computing the inner product of $\mathsf{M}[\boldsymbol{p}]$ with the row vectors mapped from $x_i, i \in [n]$. According to the computation of $\boldsymbol{p}$, for each $y_{i'} = x_i, i \in [n], i' \in [n']$, $H(x_i)/H(y_{i'})$ is equivalent to applying the corresponding row vectors of $x_i/y_{i'}$ to $\boldsymbol{p}$. Thus, given the VOLE relation of $\boldsymbol{p}$, both parties are able to encode identical elements into identical encodings. An enhanced Oblivious Key-Value Store (OKVS) scheme from [RR22a] is used here to sample the random function row, solve the linear system of $\boldsymbol{p}$, and compute the Decode algorithm. We proved that this crowd PSI protocol is maliciously secure in each distributed two-party PSI computation below. We also want to noted that the coin-flipping procedure in generating w is not mandatory in this crowd PSI protocol as it is originally used for achieving the OPRF protocol in [RS21], which requires uniformity of OPRF output.

Theorem 4. *Protocol* Π_{PSI} *in Figure 6 securely instantiated* $\mathcal{F}_{\mathsf{PSI}}$ *in* $(\mathcal{F}_{\mathsf{C-VOLE}}, H_1, H_2)$*-hybrid model against malicious adversaries.*

Proof. The proof is deferred to Section A.10 □

Additionally, even in semi-honest setting, implementing C-VOLE in crowd PSI could bring the correction vector, which is computed from the difference between randomized VOLE vector and the designated vector, to the initialization phase. This optimization ensures the same correction vector is used for all senders, reduces communication overhead by eliminating the need for individual correction vectors for each sender.

7 Performance Evaluation

We implement our proposed C-VOLE protocol and the C-VOLE based crowd PSI protocol based on the work by [RR22a]. We utilize the EMP library [WMK16] for C-VOLE implementation. For the PSI protocol, we integrate the libOT [RR] and volePSI [RR22b] libraries for Paxos operations. We instantiate all the protocols with $\mathbb{F} = \mathbb{F}_{2^{128}}$, the computational security parameter $\kappa = 128$, and the statistical security parameter $\lambda = 40$. All experiments were executed on AWS EC2 instances `6a.8xlarge` with 32vCPU and 128GB memory. We tested our schemes in both the LAN and WAN networks, where we set the LAN network with 0 ms latency and 1Gbps bandwidth, and WAN network with 400 ms latency and 50 Mbps bandwidth.

7.1 Parameter Selection of Hybrid Commitment

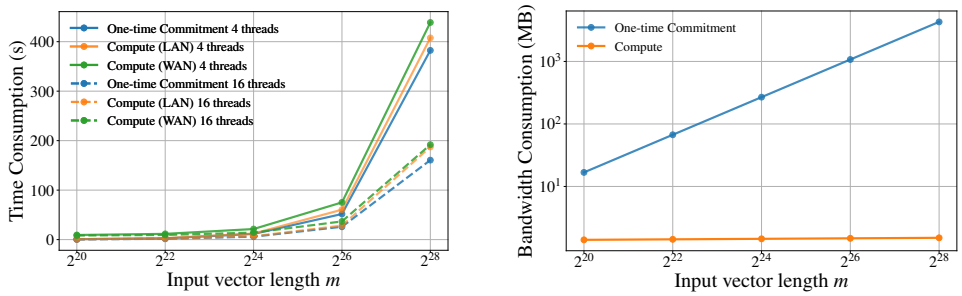
We explain how to choose parameters for a hybrid commitment $\mathsf{Hcom}_{\boldsymbol{x}}$ of $\boldsymbol{x} \in \mathbb{F}^m$, which will influence the performance of C-VOLE as well. Notice that $\mathsf{Hcom}_{\boldsymbol{x}} = \{\mathsf{com}_{\boldsymbol{e}_u}, \boldsymbol{x}^* = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x}\}$ consists of two parts: the LPN-based commitment $\mathsf{com}_{\boldsymbol{e}_u}$ of $\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}$; and a pseudorandom correction vector $\boldsymbol{x}^*$ corrects $\boldsymbol{x}$ from $\boldsymbol{e}_u$. We first select the parameters used for computing $\boldsymbol{x}^*$, which hides $\boldsymbol{x}$ with 128-bit security. Our framework is compatible with any LPN coding in the VOLE context [BCG+22, RRT23]. We use the expand-accumulate (EA) code [BCG+22] to build the matrix $\mathbf{H} \in \mathbb{F}^{w \cdot 2^L \times m}$, where the expansion parameter C corresponds to the "expand" step in EA coding. This matrix compresses vector $\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}$ into a pseudorandom vector of length m. Following the conservative constraints in Section 7.1 and [BCG+22], we set the expansion parameter $C = 10$ and the code rate $R = 1/7$ (i.e., $7 * |\boldsymbol{x}| = |\boldsymbol{e}_u|$). Under these settings, we choose the (w, L) pair for $\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}$ to achieve 128-bit security. The chosen parameters are shown in Table 1. Notice that [RRT23]'s attack only applies to the aggressive parameters of EA code but not the conservative ones we use.[1]

Given the (w, L) pair for m, we fix the structure of $\boldsymbol{e}_u$ for each $\boldsymbol{x}$. Then, we choose parameters for constructing a commitment $\mathsf{com}_{\boldsymbol{e}_u}$ with 128-bit security level, where $\mathsf{com}_{\boldsymbol{e}_u} = \boldsymbol{e}_u \cdot \mathbf{H}_1 + \boldsymbol{e}_r \cdot \mathbf{H}_2$ for the commitment from the dual LPN assumption, to achieve statistically binding and computationally hiding commitment of $\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}$. According to Theorem 1, we select the (n, t, L') pair for dual LPN commitment. For each $\boldsymbol{x} \in \mathbb{F}^m$ of different lengths, we compute groups of parameters (w, L, n, t, L') and select the one that minimizes the communication cost of the C-VOLE protocol from the dual LPN assumption, mentioned in Section 5.2 for implementation. We give the commitment parameter selection that achieving 128 bit security for $\boldsymbol{x} \in \{\mathbb{F}^{2^{20}}, \mathbb{F}^{2^{24}}, \mathbb{F}^{2^{28}}\}$ in Table 1.

[1] The attack only applies to $\mathbb{F}_2$ while our implementation is based on $\mathbb{F}_2^{128}$. Therefore, we also provide performance evaluation of C-VOLE under aggressive parameters in Section A.4.

Input	Intermediate sparse vector			Dual LPN commitment			
$\lvert \boldsymbol{x} \rvert$	$\lvert \boldsymbol{e}_u \rvert$	w	L	$\lvert \boldsymbol{e}_r \rvert$	t	L'	$\lvert \mathsf{com}_{\boldsymbol{e}_u} \rvert(n)$
2^{20}	$7 * 2^{20}$	224	15	1456	91	4	696
2^{24}	$7 * 2^{24}$	224	19	1472	92	4	710
2^{28}	$7 * 2^{28}$	224	23	1488	93	4	723

Table 1: Dual-LPN-based hybrid commitment parameters selection of $\boldsymbol{x}$ with different lengths. Achieves 128-bit security.



(a) Time consumption in different network.

(b) Bandwidth consumption.

Figure 7: **Performance of C-VOLE protocol in LAN and WAN setting.**

7.2 Efficiency of Committed-VOLE

Our C-VOLE protocol includes two phases: computing a one-time commitment on the VOLE sender's side ($\mathsf{P_A}$ in the protocol) for chosen input vector $\boldsymbol{x}$ and a compute phase that can be executed between the sender and each individual VOLE receiver ($\mathsf{P_B}$ in the protocol). We show the time consumption of the two phases under LAN and WAN with different input vector lengths m and different threads in Figure 7a. The communication overhead is shown in Figure 7b. The detailed analysis is as follows.

One-time Commitment: In this phase, the sender, who holds the input vector $\boldsymbol{x}$, generates an LPN-based hybrid commitment of $\boldsymbol{x}$ and publishes it to all the parties. We tested this process using 4 threads and 16 threads. The results show that it takes 0.84 s for $m = 2^{20}$ and 382.39 s for $m = 2^{28}$ using 4 threads, while it takes 0.33 s for $m = 2^{20}$ and 160.62 s for $m = 2^{28}$ using 16 threads. The communication overhead in this phase is publishing the hybrid commitment of $\boldsymbol{x}$, which costs 16.78 MB for $m = 2^{20}$ and 4294.9 MB for $m = 2^{28}$, dominating the entire communication of the C-VOLE protocol. As the one-time commitment is published and applicable to each individual receiver interacting with the sender to generate C-VOLE of $\boldsymbol{x}$, the time consumption for the computation can be amortized to a very small portion of the overall computation. Additionally, the commitment computation is computed locally on the sender's side and is unaffected by network settings.

Compute: We tested the compute phase for the sender generating VOLE of the committed $\boldsymbol{x}$ with one receiver. Note that the performance of the compute phase is influenced by the parameters chosen in Section 7.1, specifically the Hamming weight and the block size of the sparse vectors in executing Π_{MPVOLE}. For $m = 2^{20}$, the compute phase takes 0.81 s under a LAN network and 9.31 s under a WAN network with a bandwidth of 50 Mbps when using 4 threads. With 16 threads, the compute phase takes 0.62 seconds under LAN network and 8.20 seconds under WAN network. For $m = 2^{28}$, the compute phase with 4 threads takes 407.6 s under a LAN network and 438.8 s under a WAN network. With 16 threads, the corresponding times are 187.22 s under a

Schemes	50 Mbps	200 Mbps	500 Mbps	1 Gbps
C2A [WYX+21]	45μs	45μs	44μs	44μs
C-VOLE	1.61μs	1.59μs	1.57μs	1.53μs

Table 2: Time consumption compared with [WYX+21]. With different network settings, we tested C-VOLE of 128-bit data block to compare with C2A of 64-bit data block. Both evaluations are with single thread.

Schemes	2^{20} block	2^{24} block	2^{28} block
C2A [WYX+21]	114.52 MB	1829.42 MB	29317.12 MB
C-VOLE	18.49 MB	270.18 MB	4296.74 MB

Table 3: Bandwidth consumption compared with [WYX+21]. we tested the bandwidth consumption of C-VOLE and C2A under varying input blocks. For each C-VOLE input block, we used a 128-bit data block, while for C2A, a 64-bit data block was employed for comparison.

LAN network and 191.82 s under a WAN network. In terms of communication overhead, which is dominated by the cost of Π_{MPVOLE}, the cost is 1.72 MB for $m = 2^{20}$ and 1.78 MB for $m = 2^{28}$. The communication overhead does not increase significantly with the increase of m, benefiting from the sublinear online communication of our intermediate structured sparse LPN commitment and corresponding consistency check construction.

7.3 Comparison with Prior Work

7.3.1 Comparison with C2A Conversion.

We compare the proposed C-VOLE protocol with the public commitment to private authentication conversion (C2A) in [WYX+21], which is used for commit-and-prove zero-knowledge proofs. The C2A conversion in [WYX+21] assumes 64-bit data block, while our protocol is evaluated under field $\mathbb{F}_{2^{128}}$, implemented as data block of 128-bit. The time consumption and communication overhead are summarized in Table 2 and Table 3 and are analyzed below.

Wallclock time. C2A conversion is the bottleneck of computing commit-and-prove ZK. As tested from [WMK16], the C2A takes 45 μs per element while the zero-knowledge proof only takes less than 0.1 μs per gate [YSWW21]. We tested ours C-VOLE computation time for each block under different network setting, each result is amortized from the performance of vector of length 2^{24}. The results show that our scheme requires 1.61μs computation for 128-bit data block, which is 28× faster than the 45μs computation for 64-bit data block in [WYX+21]. Note that a follow-up LPZKv2 [DILO22b] provides at most a 2× improvement over QuickSilver [YSWW21]. The advantage of C-VOLE compared to Mystique remains significant even if its underlying ZKP is replaced from QuickSilver to LPZKv2.

Communication overhead. We tested the communication overhead of the C2A conversion and our proposed C-VOLE for input vector length $m = \{2^{20}, 2^{24}, 2^{28}\}$. The communication overhead includes the published commitment, the interactive generation of VOLE, and the proof of input vector consistency. C2A conversion requires 114.52 MB of communication for 2^{20} blocks while ours C-VOLE only requires 18.49 MB. For 2^{28} blocks, the C2A consumes 29 GB communication overhead and ours C-VOLE consumes 4.2 GB. Note that in the C2A conversion, the commitment size is small, and the proof of consistency dominates the entire communication cost. In contrast, for our C-VOLE, the commitment size constitutes the majority of the communication overhead.

This means that when executing C2A operations multiple times on the same publicly committed data, ours C-VOLE protocol is much more communication-efficient.

7.3.2 Comparison with CVOLE with Input-revealing Identifiable-abort (IRIA).

We compare the committing VOLE proposed in [CDKs24] with our proposed scheme. The committing VOLE with input-revealing identifiable-abort is designed having both parties commit to their input and reveal them to prove consistency in case of an abort. To ensure a fair comparison with our work, we only require the sender to commit to its input vector and reveal it for consistency checks. Since there is no implementation available for [CDKs24], we analyze and compare its communication overhead with ours theoretically.

The C-VOLE used in [CDKs24] can be divided into two steps according to their paper: 1) Generate oblivious transfer (OT) based on PVW OT [PVW08] and extend it to VOLE using OT-multiplication protocols [DKLs18, Gil99]; 2) Run a sigma protocol to reveal the input to the other party and prove consistency of the input with previous transferred messages for each OT. For step 1), each DDH-based PVW OT requires $8 \log |\mathbb{G}|$ communication. To extend it to VOLE with vector length m, the extension [DKLs18] requires $m \cdot (\log |\mathbb{F}| + \kappa + 2\lambda)$ execution of OTs, along with additional consistency check of the sender's input for malicious security, which doubles the the number of OTs. For step 2), each OT requires a communication overhead of $4 \log |\mathbb{F}| + 8 \log |\mathbb{G}|$ for the sigma protocol to prove consistency with the commitment (transferred message). The total communication overhead for the C-VOLE with sender-committed input and revealing is $2 \cdot m \cdot (\log |\mathbb{F}| + \kappa + 2\lambda) \cdot (8 \log |\mathbb{G}| + 4 \log |\mathbb{F}| + 8 \log |\mathbb{G}|)$. In contrast, ours communication overhead is $(wL + tL')\kappa + (n + m) \log |\mathbb{F}|$ as analyzed in Section 5.2. For $m = 2^{20}$, $\mathbb{G}$ with order 128 bits, the communication cost for [CDKs24] is 225 GB, whereas ours is 16.84 MB.

7.4 Efficiency of C-VOLE based PSI and Comparison

In this section, we evaluate the performance of our proposed C-VOLE based crowd PSI and compare it with the state-of-the-art two party PSI [RR22a] and the state-of-the-art crowd PSI protocol [SKR+24].

We integrate C-VOLE into VOLE-based PSI protocol, Blazing PSI [RR22a], which is also the state-of-the-art two-party PSI protocol, by replacing VOLE with C-VOLE to achieve malicious security in the crowd setting. As shown in Figure 6, in the C-VOLE based crowd PSI protocol, the C-VOLE sender acts as the receiver (server) of the crowd PSI protocol and computes a publicly committed vector $\boldsymbol{p}$ for initialization. The server (receiver) then computes the extend phase of crowd PSI with each individual client (sender) to get the intersection. The initialization phase includes a Paxos encoding operation and the computation of a one-time commitment. The extend phase consists of the following steps: 1) Both the server and the client generate the VOLE correlation of the committed $\boldsymbol{p}$; 2) The client sends the encoding of its set elements to the server; 3) The server performs local encoding of its set and compares the two encodings to determine the intersection with each client.

7.4.1 Comparison with VOLE based PSI.

The only difference between ours C-VOLE based PSI and the VOLE-based PSI [RR22a] is that we use committed-VOLE instead of VOLE. Notice that [RR22a] proposed two versions of implementation, one uses VOLE with slightly faster computational overhead, the other one uses subfield-VOLE with reduced communication overhead. We compare with the faster version (VOLE-based) and measure both the computation and the communication overhead by fixing client's set to 2^{10}

elements and scaling the server's set to $\{2^{20}, 2^{22}, 2^{24}\}$. The detailed comparison are presented in Table 4 and Table 5.

Set size (m, n)		$(2^{20}, 2^{10})$	$(2^{22}, 2^{10})$	$(2^{24}, 2^{10})$
Blazing PSI [RR22a] (s)		0.41	2.44	9.89
C-VOLE PSI	Init (s)	0.78	3.34	12.37
	Online (s)	1.08	3.51	14.69

Table 4: Time consumption compared with [RR22a]. (m, n) represents the server's set size m, and client's set size n. PSI Init is a one-time computation and PSI online is executed between the server and each individual client. The experiments are tested with 16 threads.

Set size (m, n)		$(2^{20}, 2^{10})$	$(2^{22}, 2^{10})$	$(2^{24}, 2^{10})$
Blazing PSI [RR22a] (MB)		23.83	92.34	366.55
C-VOLE PSI	Init (MB)	21.94	88.41	353.94
	Online (MB)	1.73	1.75	1.76

Table 5: Communication overhead compared with [RR22a]. (m, n) represents the server's set size m, and client's set size n. The experiments are executed with 16 threads.

Compared with Blazing PSI [RR22a], the results show that our C-VOLE based PSI consumes approximately $2\times$ more online running time but requires significantly less online communication overhead. This difference can be attributed to the fact that Blazing PSI employs the default VOLE implementation in the library LibOTe using EC code. In contrast, ours uses EA code with parameter $R = \frac{1}{7}$, $C = 10$ and specifically selects the structure of $\boldsymbol{e}_u$ and $\boldsymbol{e}_r$ for both security and performance in Section 7.1. Moreover, our C-VOLE based PSI has the property of having the initialization phase performed locally in advance to enhance performance. In the crowd PSI setting, when a server executes PSI with multiple clients, the initialization can be amortized to a very small amount of time. The communication can also be managed by other efficient networks such as CDN or P2P, further enhancing performance.

7.4.2 Comparison with Prior Work in Crowd PSI.

There's only one prior work that considers PSI in the crowd setting [SKR+24]. In this work, the server commits to its private input and executes PSI with each individual client using an oblivious verifiable unpredictable function (OVUF). This work allows the client to obtain the intersection, whereas our protocol enables the server to obtain the intersection. We compare our work with this work despite the slight difference in output, as it provides the relevant baseline for performance in crowd setting. We set the server's set size to $\{2^{20}, 2^{24}, 2^{28}\}$ elements and the client's set size to $\{2^{10}, 2^{17}, 2^{20}\}$ elements, and present the performance results below.

Wallclock time. We evaluate the time consumption for executing PSI between a server and a client using 16 threads. As shown in Table 6, the initialization of both schemes scales linearly with the server's set size, as it computes commitment for the server's private set elements. Ours C-VOLE PSI requires 465.23 ms for a server with 2^{28} elements, while OVUF-PSI requires 1596.44 ms. For the online phase, our C-VOLE based PSI includes generating the VOLE correlation of the committed vector (with a length linear to the server set size) and computing the client's side encoding to the server. The time consumption scales roughly linearly with the server's set size, with a small additional increase as the client set size grows. In contrast, the OVUF-PSI scales

Set size (m, n)	C-VOLE PSI		OVUF-PSI [SKR+24]	
	Init (s)	Online (s)	Init (s)	Online (s)
$(2^{20}, 2^{10})$	0.78	1.03	6.24	0.59
$(2^{20}, 2^{17})$	0.79	1.08	6.24	67.36
$(2^{20}, 2^{20})$	0.79	1.23	6.24	538.88
$(2^{24}, 2^{10})$	12.37	14.35	98.14	0.59
$(2^{24}, 2^{17})$	12.37	14.69	98.14	67.36
$(2^{24}, 2^{20})$	458.17	455.55	98.14	538.88
$(2^{28}, 2^{10})$	458.10	499.04	1596.44	0.59
$(2^{28}, 2^{17})$	465.23	523.17	1596.44	67.36
$(2^{28}, 2^{20})$	80.82	87.16	1596.44	538.88

Table 6: Time consumption compared with [SKR+24]. (m, n) represents the server's set size m, and the client's set size n. PSI Init is a one-time computation and PSI Online is executed between the server and each individual client. All experiments are executed with 16 threads. The online computation time for [SKR+24] with a client size of 2^{20} is derived from other tested results based on linear properties.

linearly with the client set size. For a server's set size of 2^{20}, ours C-VOLE PSI requires 1.03 s for client size 2^{10} and 1.23 s for client size 2^{20}. In comparison, OVUF-PSI requires 0.59 s for client size 2^{10} and 538.88 s for client size 2^{20}.

Communication overhead. For communication overhead, the initialization phase of both schemes requires the server to publish the commitment of its set elements. The communication overhead scales linearly with the server set size. However, in OVUF-PSI, each element in $\mathbb{F}_{2^{128}}$ is hashed to 64-bit, while ours C-VOLE PSI can not use such hashing. The communication overhead of our initialization phase scales linearly with the length of $\boldsymbol{p}$, which is proportional to the server's set size. Consequently, our C-VOLE PSI requires roughly 2.6 times more communication overhead than the OVUF-PSI during initialization. In the online phase, C-VOLE PSI includes generating the C-VOLE correlation of $\boldsymbol{p}$ and having the client send back its set encodings. The online communication overhead is dominated by the client's set size, requires 1.73 MB for a client size of 2^{10} and 18.49 MB for a client size of 2^{20}, with a fixed server size 2^{20}. This significantly outperforms the OVUF-PSI online phase.

Set size (m, n)	C-VOLE PSI		OVUF-PSI [SKR+24]	
	Init (MB)	Online (MB)	Init (MB)	Online (MB)
$(2^{20}, 2^{10})$	21.94	1.73	8.38	63.23
$(2^{20}, 2^{17})$	21.94	3.81	8.38	8091
$(2^{20}, 2^{20})$	21.94	18.49	8.38	64747
$(2^{24}, 2^{10})$	353.94	1.76	134.18	63.23
$(2^{24}, 2^{17})$	353.94	3.84	134.18	8091
$(2^{24}, 2^{20})$	353.94	18.52	134.18	64747
$(2^{28}, 2^{10})$	5671.63	1.79	2147	63.23
$(2^{28}, 2^{17})$	5671.63	3.87	2147	8091
$(2^{28}, 2^{20})$	5671.62	18.55	2147	64747

Table 7: Communication overhead compared with [SKR+24]. (m, n) represents the server's set size m, and client's set size n. The online communication time for [SKR+24] with a client size of 2^{20} is derived from other tested results based on linear properties.

Acknowledgements

The work of Yu Yu is supported by the National Key Research and Development Program of China (Grant No. 2020YFA0309705), National Natural Science Foundation of China (Grant Nos. 62125204 and 92270201), Innovation Program for Quantum Science and Technology (No. 2021ZD0302901/2021ZD0302902), and the Major Program of Guangdong Basic and Applied Research (Grant No. 2019B030302008). The work of Xiao Wang is supported by NSF award #2236819. Yu Yu's work has also been supported by the New Corner-stone Science Foundation through the XPLORER PRIZE.

References

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symm. Cryptol.*, 2020(3):1–45, 2020.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219, Hanoi, Vietnam, December 4–8, 2016. Springer Berlin Heidelberg, Germany.

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2105–2122, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer Berlin Heidelberg, Germany.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 278–291, Santa Barbara, CA, USA, August 22–26, 1994. Springer Berlin Heidelberg, Germany.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303, Vienna, Austria, October 24–28, 2016. ACM Press.

[BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[CDKs24] Ran Cohen, Jack Doerner, Yashvanth Kondi, and abhi shelat. Secure multiparty computation with identifiable abort via vindicating release. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 36–73, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[DILO22a] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[DILO22b] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Improving line-point zero knowledge: Two multiplications for the price of one. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 829–841, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In Stefano Tessaro, editor, *ITC 2021*, volume 199 of *LIPIcs*, pages 5:1–5:24, Seattle, WA, USA, July 23–26, 2021. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.

[EPSW24] Daniel Escudero, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. Multi-verifier zero-knowledge proofs for any constant fraction of corrupted verifiers. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 4092–4106, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

[Gil99] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129, Santa Barbara, CA, USA, August 15–19, 1999. Springer Berlin Heidelberg, Germany.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 11–13, 2021.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the poseidon hash function. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 177–203, Sousse, Tunisia, July 19–21, 2023. Springer, Cham, Switzerland.

[HSS20] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. *Journal of Cryptology*, 33(4):1732–1786, October 2020.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966, Berlin, Germany, November 4–8, 2013. ACM Press.

[JKPT12] Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 663–680, Beijing, China, December 2–6, 2012. Springer Berlin Heidelberg, Germany.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378, Santa Barbara, CA, USA, August 16–20, 1988. Springer Berlin Heidelberg, Germany.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany.

[RR] Peter Rindal and Lance Roy. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[RR22a] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2505–2517, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[RR22b] Srinivasan Raghuraman and Peter Rindal. VOLE-PSI. https://github.com/Visa-Research/volepsi, 2022.

[RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 602–632, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.

[RS22] Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072, London, UK, November 11–15, 2019. ACM Press.

[SKR+24] Yunqing Sun, Jonathan Katz, Mariana Raykova, Phillipp Schoppmann, and Xiao Wang. Actively secure private set intersection in the client-server setting. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1478–1492, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 39–56, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[WYX+21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 501–518. USENIX Association, August 11–13, 2021.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1607–1626, Virtual Event, USA, November 9–13, 2020. ACM Press.

A Appendix

A.1 Hybrid Commitment Parameters based on EA-Code

Boyle et al. [BCG+22] introduced a method of choosing conservative parameters for EA-code, which is secure against the current known attacks [RRT23]. They use a random walk on a Markov chain to determine the minimum distance for the EA code. As all variants of LPN generalize naturally to larger fields [BCG+22], we instantiate our commitment of $\mathbb{F}_{2^{128}}$ based on the conservative parameters constraints of $\mathbb{F}_2$ as follows.

A.1.1 Conservative parameters

Based on [BCG+22, Theorem 3.10], for $\mathbf{H} \leftarrow \mathrm{EAGen}(n, N, \frac{C \ln N}{N})$, $\mathbf{H}$ achieves a minimum distance of δN with a probability of at least

$$\eta(\kappa) \geq 1 - 2RN^{-2\beta^2 C+2} = 1 - 2nN^{-2(1/2-\delta)^2 C+1} \ . \tag{1}$$

For an error distribution with an expected weight of t, we have

$$\varepsilon_d \leq (1 - 2t/N)^{\delta N}/2 \leq e^{-2t\delta} \ . \tag{2}$$

We consider two scenarios: a failure probability of at most 0.01 (i.e., $\eta \geq 0.99$). Our chosen failure probability falls within the range of $[0.0000148, 0.174]$, as indicated in the initial work [BCG+22, Figure 8], and is deemed reasonable. To achieve 128-bit security against linear tests, we have $-\log(\varepsilon_d) = 128 - \log(N)$ because computing a dot-product requires N operations [BCG+22]. We first choose an appropriate value for t and then determine δ using Equation (2), which is $\delta \geq \frac{128-\log(N)}{2\log(e)\cdot t}$. Next, to ensure $\eta \geq 0.99$, we use Equation (1) and choose C such that

$$\log(2n) + \log(N) \cdot (-2(1/2 - \delta)^2 C + 1) \leq \log(0.01) \ .$$

Finally, we check the parameters to satisfy the following conditions from [BCG+22, Theorem 3.10]:

$$R < \min\left\{\frac{2}{\ln 2} \cdot \frac{1 - e^{-1}}{1 + e^{-1}} \cdot \beta^2, \frac{2}{e}\right\} \quad \text{and} \quad C > \frac{1}{\beta^2} \ .$$

shown in Theorem [BCG+22, Theorem 3.10], where $\beta = 1/2 - \delta$.

A.1.2 Aggressive parameters

[BCG+22] proposed aggressive parameters by empirically estimating the average minimum row-weight of $\mathbf{H} \leftarrow \mathrm{EAGen}(n, N, \ell)$ using exponential regression in $\log(n)$, with $N = 5n$ and $\ell \in \{7, 9, 11\}$. Specifically, $\mathbf{H}$ achieves a minimum distance of δN, where

$$\delta \approx \begin{cases} 0.4646 \cdot e^{-0.1012 \cdot \log(n)}, & \text{if } \ell = 7 \\ 0.4415 \cdot e^{-0.0783 \cdot \log(n)}, & \text{if } \ell = 9 \\ 0.4337 \cdot e^{-0.06383 \cdot \log(n)}, & \text{if } \ell = 11 \end{cases}.$$

Then, we determine t using Equation (2), which is $t \geq \frac{128 - \log(N)}{2\log(e) \cdot \delta}$.

A.2 Primal-LPN based C-VOLE

We give primal-LPN based C-VOLE construction in this section. It follows the same idea in Section 5, except minor changes regarding the format of primal-LPN commitment of vector $\boldsymbol{e}_u$. As we introduced, generating any C-VOLE of designated vector $\boldsymbol{x}$ is start with generating C-VOLE of a designated regular-w vector $\boldsymbol{e}_u$. As $\boldsymbol{e}_u$ is committed as $\mathsf{com}_{\boldsymbol{e}_u} = \boldsymbol{e}_u \cdot \mathbf{A}_1 + \boldsymbol{r} \cdot \mathbf{A}_2 + \boldsymbol{e}_r$ and $\mathsf{decom}_{\boldsymbol{e}_u} = \{\boldsymbol{r}, \boldsymbol{e}_r\}$ under primal-LPN commitment in Section 4.1, we generate consistent $[\![\boldsymbol{e}_u]\!]$ as follows: 1) generate $[\![\boldsymbol{e}_u]\!]$ with $\mathsf{P_A}$ input $\boldsymbol{e}_u$, 2) generate $[\![\boldsymbol{e}_r]\!]$ with $\mathsf{P_A}$ input $\boldsymbol{e}_r$, 3) generate $[\![\boldsymbol{r}]\!]$ with $\mathsf{P_A}$ input $\boldsymbol{r}$, 4) $\mathsf{P_A}$ computes $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])$ and sends it to $\mathsf{P_B}$ for consistency check. The consistency check works as follows:

$$\mathsf{M}[\mathsf{com}_{e_u}] = \mathsf{M}[\boldsymbol{e}_u] \cdot \mathbf{A}_1 + \mathsf{M}[\boldsymbol{r}] \cdot \mathbf{A}_2 + \mathsf{M}[\boldsymbol{e}_r]$$
$$\mathsf{K}[\mathsf{com}_{e_u}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{A}_1 + \mathsf{K}[\boldsymbol{r}] \cdot \mathbf{A}_2 + \mathsf{K}[\boldsymbol{e}_r]$$

If $[\![\boldsymbol{e}_u]\!]$, $[\![\boldsymbol{e}_r]\!]$, and $[\![\boldsymbol{r}]\!]$ are generated with $\mathsf{P_A}$ input consistent $\boldsymbol{e}_u$, $\boldsymbol{e}_r$, and $\boldsymbol{r}$ used in $\mathsf{com}_{\boldsymbol{e}_u}$, $[\![\mathsf{com}_{\boldsymbol{e}_u}]\!]$ should hold and be checked successfully by $\mathsf{P_B}$ that $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]) = H_3(\mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}] + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta)$.

Then, we can recover $[\![\boldsymbol{x}]\!]$ from $[\![\boldsymbol{e}_u]\!]$ under $\mathsf{Hcom}_{\boldsymbol{x}}$ as easily as dual-LPN-based C-VOLE does in Section 5. The detailed protocol is shown in Figure 8. We also prove that the protocol is secure against malicious adversaries in Theorem 5.

Theorem 5. *Protocol $\Pi_{\mathsf{C-VOLE-primal}}$ in Figure 8 securely instantiated $\mathcal{F}_{\mathsf{C-VOLE}}$ in Figure 1 in ($\mathcal{F}_{\mathsf{MPVOLE}}$,$\mathcal{F}_{\mathsf{VOLE}}$)-hybrid model against any malicious adversaries.*

Proof. Let $\mathcal{A}$ be a PPT adversary that corrupt $\mathsf{P_A}$ or $\mathsf{P_B}$. We construct a PPT simulator $\mathcal{S}$ with access to $\mathcal{F}_{\mathsf{C-VOLE}}$ and simulates the adversary's view. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the output of $\mathcal{S}$ and the honest party in the ideal world.

Corrupted $\mathsf{P_A}$. Let $\mathcal{S}$ access $\mathcal{F}_{\mathsf{C-VOLE}}$ as an honest $\mathsf{P_A}$ and interact with $\mathcal{A}$ as an honest $\mathsf{P_B}$. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

1. $\mathcal{S}$ holds public commitment of $\boldsymbol{x}$, that $\mathsf{Hcom}_{\boldsymbol{x}} = (\mathsf{com}_{\boldsymbol{e}_u}, \boldsymbol{x}^*)$. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives (init) from $\mathcal{A}$.

2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives $(\mathsf{extend}, \boldsymbol{e}'_u \in \mathbb{F}^{w \cdot 2^L}, L, w)$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{M}[\boldsymbol{e}'_u] \in \mathbb{F}^{w \cdot 2^L}$ from $\mathcal{A}$. $\mathcal{S}$ computes $\boldsymbol{x}' = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x}^*$.

3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives $(\mathsf{extend}, \boldsymbol{e}'_r \in \mathbb{F}^{t \cdot 2^{L'}}, L', t)$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{M}[\boldsymbol{e}'_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ from $\mathcal{A}$.

Protocol $\Pi_{\mathsf{C-VOLE-primal}}$

Parameters: A public hybrid commitment of $\boldsymbol{x} \in \mathbb{F}^m$ defined in Section 4.2, that $\mathsf{Hcom}_{\boldsymbol{x}} = (\mathsf{com}_{\boldsymbol{e}_u} \in \mathbb{F}^N, \boldsymbol{x}^* = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x} \in \mathbb{F}^m)$. A primal LPN commitment scheme defined in Section 4.1 that $(\mathsf{com}_{\boldsymbol{e}_u} = \boldsymbol{e}_u \cdot \mathbf{A}_1 + \boldsymbol{r} \cdot \mathbf{A}_2 + \boldsymbol{e}_r, \mathsf{decom}_{\boldsymbol{e}_u} = \{\boldsymbol{r}, \boldsymbol{e}_r\}) \leftarrow \mathsf{Com}(\boldsymbol{e}_u)$, where $\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}$, $\boldsymbol{r} \in \mathbb{F}^k$, $\boldsymbol{e}_r \in \mathbb{F}^{t \cdot 2^{L'}}$, $N = t \cdot 2^{L'}$. A Collision-Resistant Hash Function H_3.

Inputs: $\mathsf{P_A}$ holds vector $\boldsymbol{x} \in \mathbb{F}^m, \mathsf{Hdecom}_{\boldsymbol{x}} = (\boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}, \boldsymbol{r} \in \mathbb{F}^k, \boldsymbol{e}_r \in \mathbb{F}^{t \cdot 2^{L'}})$.

Compute:

1. Both $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{MPVOLE}}$ with input (init). $\mathsf{P_B}$ receives $\Delta \in \mathbb{F}$ from $\mathcal{F}_{\mathsf{MPVOLE}}$.
2. $\mathsf{P_A}$ calls $\mathcal{F}_{\mathsf{MPVOLE}}$ with input $(\mathsf{extend}, \boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}, L, w)$. $\mathsf{P_B}$ calls $\mathcal{F}_{\mathsf{MPVOLE}}$ with input (extend, L, w). The functionality returns $\mathsf{M}[\boldsymbol{e}_u] \in \mathbb{F}^{w \cdot 2^L}$ to $\mathsf{P_A}$ and $\mathsf{K}[\boldsymbol{e}_u] \in \mathbb{F}^{w \cdot 2^L}$ to $\mathsf{P_B}$ such that $\mathsf{M}[\boldsymbol{e}_u] = \mathsf{K}[\boldsymbol{e}_u] + \Delta \cdot \boldsymbol{e}_u \in \mathbb{F}^{w \cdot 2^L}$. If either party receives abort from $\mathcal{F}_{\mathsf{MPVOLE}}$ in any of these executions, it aborts.
3. $\mathsf{P_A}$ calls $\mathcal{F}_{\mathsf{MPVOLE}}$ with input $(\mathsf{extend}, \boldsymbol{e}_r \in \mathbb{F}^{t \cdot 2^{L'}}, L', t)$. $\mathsf{P_B}$ calls $\mathcal{F}_{\mathsf{MPVOLE}}$ with input (extend, L', t). The functionality returns $\mathsf{M}[\boldsymbol{e}_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ to $\mathsf{P_A}$ and $\mathsf{K}[\boldsymbol{e}_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ to $\mathsf{P_B}$ such that $\mathsf{M}[\boldsymbol{e}_r] = \mathsf{K}[\boldsymbol{e}_r] + \Delta \cdot \boldsymbol{e}_r \in \mathbb{F}^{t \cdot 2^{L'}}$. If either party receives abort from $\mathcal{F}_{\mathsf{MPVOLE}}$ in any of these executions, it aborts.
4. $\mathsf{P_A}$ calls $\mathcal{F}_{\mathsf{VOLE}}$ with input $(\mathsf{extend}, \boldsymbol{r} \in \mathbb{F}^k, k)$; $\mathsf{P_B}$ calls $\mathcal{F}_{\mathsf{VOLE}}$ with input (extend, k). The functionality returns $\mathsf{M}[\boldsymbol{r}] \in \mathbb{F}^k$ to $\mathsf{P_A}$ and $\mathsf{K}[\boldsymbol{r}] \in \mathbb{F}^k$ to $\mathsf{P_B}$, such that $\mathsf{M}[\boldsymbol{r}] = \mathsf{K}[\boldsymbol{r}] + \boldsymbol{r} \cdot \Delta$. If either party receives abort from $\mathcal{F}_{\mathsf{VOLE}}$ in any of these executions, it aborts.
5. $\mathsf{P_A}$ computes $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{M}[\boldsymbol{e}_u] \cdot \mathbf{A}_1 + \mathsf{M}[\boldsymbol{r}] \cdot \mathbf{A}_2 + \mathsf{M}[\boldsymbol{e}_r]$ and sends $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])$ to $\mathsf{P_B}$; $\mathsf{P_B}$ computes $\mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{A}_1 + \mathsf{K}[\boldsymbol{r}] \cdot \mathbf{A}_2 + \mathsf{K}[\boldsymbol{e}_r]$ and $\mathbf{value} = \mathsf{K}[\mathsf{com}_{\boldsymbol{e}_u}] + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta \in \mathbb{F}^N$. $\mathsf{P_B}$ checks whether $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]) = H_3(\mathbf{value})$. If not, $\mathsf{P_B}$ aborts.
6. $\mathsf{P_A}$ outputs $(\boldsymbol{x}, \mathsf{M}[\boldsymbol{x}])$, where $\mathsf{M}[\boldsymbol{x}] = \mathsf{M}[\boldsymbol{e}_u] \cdot \mathbf{H} \in \mathbb{F}^m$. $\mathsf{P_B}$ outputs $(\mathsf{K}[\boldsymbol{x}], \Delta)$, where $\mathsf{K}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H} + \boldsymbol{x}^* \Delta \in \mathbb{F}^m$

Figure 8: The committed-VOLE protocol based on primal-LPN-based hybrid commitment

4. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ and receives $(\mathsf{extend}, \boldsymbol{r}' \in \mathbb{F}^k, k)$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{M}[\boldsymbol{r}'] \in \mathbb{F}^k$ to $\mathcal{A}$. $\mathcal{S}$ sends $(\boldsymbol{x}', \mathsf{decom}_x{}' = (\boldsymbol{e}'_u, \boldsymbol{r}', \boldsymbol{e}'_r))$ to $\mathcal{F}_{\mathsf{C-VOLE}}$. If $\mathcal{S}$ receives abort from $\mathcal{F}_{\mathsf{C-VOLE}}$, $\mathcal{S}$ aborts at step (5).

5. $\mathcal{S}$ receives $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])'$ from $\mathcal{A}$. $\mathcal{S}$ checks if $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])' = H_3(\mathsf{M}[\boldsymbol{e}'_u]\cdot\mathbf{A}_1+\mathsf{M}[\boldsymbol{r}']\cdot\mathbf{A}_2+\mathsf{M}[\boldsymbol{e}'_r])$. If not, $\mathcal{S}$ aborts.

6. $\mathcal{S}$ computes $\mathsf{M}[\boldsymbol{x}'] = \mathsf{M}[\boldsymbol{e}'_u] \cdot \mathbf{H}$ and sends $\mathsf{M}[\boldsymbol{x}']$ to $\mathcal{F}_{\mathsf{C-VOLE}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $\mathcal{F}_{\mathsf{MPVOLE}}, \mathcal{F}_{\mathsf{VOLE}}$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and $\mathcal{F}_{\mathsf{VOLE}}$. Notice that $\mathcal{S}$ does not simulate any message to $\mathcal{A}$.

Hybrid $\mathcal{H}_2$ Same as Hybrid $\mathcal{H}_1$ except $\mathcal{S}$ aborts at step (5) if $\mathcal{S}$ receives abort from $\mathcal{F}_{\mathsf{C-VOLE}}$ or the received $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])' \neq H_3(\mathsf{M}[\boldsymbol{e}'_u] \cdot \mathbf{A}_1 + \mathsf{M}[\boldsymbol{r}'] \cdot \mathbf{A}_2 + \mathsf{M}[\boldsymbol{e}'_r])$.

In hybrid $\mathcal{H}_1$, an honest $\mathsf{P_B}$ aborts at step (5) if the received $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])' \neq H_3(\mathsf{K}[\boldsymbol{e}'_u] \cdot \mathbf{A}_1 + \mathsf{K}[\boldsymbol{r}'] \cdot \mathbf{A}_2 + \mathsf{K}[\boldsymbol{e}'_r] + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta)$. There are the following abort conditions with abort probability indistinguishable from hybrid $\mathcal{H}_2$:

1) $\mathcal{A}$ uses $\boldsymbol{e}'_u \neq \boldsymbol{e}_u$, $\boldsymbol{e}'_r \neq \boldsymbol{e}_r$, and $\boldsymbol{r}' \neq \boldsymbol{r}$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{A}$ computes $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]'$ honestly from $\mathsf{M}[\boldsymbol{e}'_u]\cdot\mathbf{A}_1+\mathsf{M}[\boldsymbol{r}']\cdot\mathbf{A}_2+\mathsf{M}[\boldsymbol{e}'_r]$. Any $(\boldsymbol{e}'_u, \boldsymbol{e}'_r, \boldsymbol{r}')$ inconsistent with $(\boldsymbol{e}_u, \boldsymbol{e}_r, \boldsymbol{r})$ will result the inequality above with the probability of statistical binding failure mentioned in Theorem 1. In this hybrid, any inconsistency will be checked by the $\mathcal{F}_{\mathsf{C-VOLE}}$ with the same binding failure and result in an abort at step (5), which is indistinguishable from hybrid $\mathcal{H}_1$. 2) $\mathcal{A}$ uses $\boldsymbol{e}'_u \neq \boldsymbol{e}_u$, $\boldsymbol{e}'_r \neq \boldsymbol{e}_r$, and $\boldsymbol{r}' \neq \boldsymbol{r}$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{A}$ samples $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]'$ to $\mathcal{S}$. The consistency check passes if and only if $\mathcal{A}$ samples $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]' = \mathsf{K}[\boldsymbol{e}'_u]\cdot\mathbf{A}_1+\mathsf{K}[\boldsymbol{r}']\cdot\mathbf{A}_2+\mathsf{K}[\boldsymbol{e}'_r]+\mathsf{com}_{\boldsymbol{e}_u}\cdot\Delta$. However, since $\mathsf{K}[\boldsymbol{e}'_u]$, $\mathsf{K}[\boldsymbol{e}'_r]$, $\mathsf{K}[\boldsymbol{r}]$, and Δ are uniformly distributed over $\mathbb{F}^{w\cdot 2^L}, \mathbb{F}^{t\cdot 2^{L'}}$, $\mathbb{F}^k$ and $\mathbb{F}$, respectively, the consistency check fails with all but negligible probability. In this hybrid, given inconsistent $(\boldsymbol{e}'_u, \boldsymbol{e}'_r, \boldsymbol{r}')$ pair, $\mathcal{S}$ aborts except the failure probability of binding, which is also negligible and thus indistinguishable from hybrid $\mathcal{H}_1$. 3) $\mathcal{A}$ uses $\boldsymbol{e}'_u = \boldsymbol{e}_u$, $\boldsymbol{e}'_r = \boldsymbol{e}_r$, and $\boldsymbol{r}' = vecr$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{A}$ samples $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]'$ to $\mathcal{S}$. The consistency check passes if and only if $\mathcal{A}$ samples $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]' = \mathsf{K}[\boldsymbol{e}'_u]\cdot\mathbf{A}_1+\mathsf{K}[\boldsymbol{r}']\cdot\mathbf{A}_2+\mathsf{K}[\boldsymbol{e}'_r]+\mathsf{com}_{\boldsymbol{e}_u}\cdot\Delta$, which is equivalent to checking $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]' = \mathsf{M}[\boldsymbol{e}'_u] \cdot \mathbf{A}_1 + \mathsf{M}[\boldsymbol{r}'] \cdot \mathbf{A}_2 + \mathsf{M}[\boldsymbol{e}'_r] + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta$ in this hybrid. Thus, the abort probability is indistinguishable from hybrid $\mathcal{H}_1$.

Therefore, this hybrid is identically distributed as $\mathcal{H}_1$.

Corrupted $\mathsf{P_B}$. Let $\mathcal{S}$ access $\mathcal{F}_{\mathsf{C-VOLE}}$ as an honest $\mathsf{P_B}$ and interact with $\mathcal{A}$ as an honest $\mathsf{P_A}$. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ holds public commitment of $\boldsymbol{x}$, $\mathsf{Hcom}_{\boldsymbol{x}} = (\mathsf{com}_{\boldsymbol{e}_u}, \boldsymbol{x}^* = \boldsymbol{e}_u \cdot \mathbf{H} - \boldsymbol{x})$. $\mathcal{S}$ sends (init) to $\mathcal{F}_{\mathsf{C-VOLE}}$. $\mathcal{S}$ receives $\Delta \in \mathbb{F}$ from $\mathcal{A}$.

2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives (extend, L, w) from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{K}[\boldsymbol{e}_u] \in \mathbb{F}^{w\cdot 2^L}$ from $\mathcal{A}$.

3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and receives (extend, L', t) from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{K}[\boldsymbol{e}_r] \in \mathbb{F}^{t\cdot 2^{L'}}$ from $\mathcal{A}$.

4. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ and receives (extend, k) from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{K}[\boldsymbol{r}] \in \mathbb{F}^k$ from $\mathcal{A}$.

5. $\mathcal{S}$ computes $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]) = H_3(\mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{A}_1 + \mathsf{K}[\boldsymbol{r}] \cdot \mathbf{A}_2 + \mathsf{K}[\boldsymbol{e}_r] + \mathsf{com}_{\boldsymbol{e}_u} \cdot \Delta)$ and sends it to $\mathcal{A}$. $\mathcal{S}$ aborts if $\mathcal{A}$ aborts.

6. $\mathcal{S}$ computes $\mathsf{K}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{e}_u] \cdot \mathbf{H} + \boldsymbol{x}^* \cdot \Delta \in \mathbb{F}^m$ and sends $(\Delta, \mathsf{K}[\boldsymbol{x}]) \in \mathbb{F}^{m+1}$ to $\mathcal{F}_{\mathsf{C-VOLE}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

Input	Random sparse vector			Primal LPN commitment				
$\lvert\boldsymbol{x}\rvert$	$\lvert\boldsymbol{e}_u\rvert$	w	L	$\lvert\boldsymbol{e}_r\rvert$	t	L'	$\lvert\boldsymbol{r}\rvert(k)$	$\lvert\mathsf{com}_{\boldsymbol{e}_u}\rvert$
2^{20}	$7*2^{20}$	224	15	1272	159	3	433	1272
2^{24}	$7*2^{24}$	224	19	1296	162	3	434	1296
2^{28}	$7*2^{28}$	224	23	1296	162	3	434	1296

Table 8: Primal-LPN-based hybrid commitment parameters selection of $\boldsymbol{x}$ with different lengths. Achieves 128-bit security.

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $\mathcal{F}_{\mathsf{MPVOLE}}, \mathcal{F}_{\mathsf{VOLE}}$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPVOLE}}$ and $\mathcal{F}_{\mathsf{VOLE}}$, and simulate messages to $\mathcal{A}$.

$\mathcal{S}$ simulates $H_3(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]) = H_3(\mathsf{K}[\boldsymbol{e}_u]\cdot\mathbf{A}_1+\mathsf{K}[\boldsymbol{r}]\cdot\mathbf{A}_2+\mathsf{K}[\boldsymbol{e}_r]+\mathsf{com}_{\boldsymbol{e}_u}\cdot\Delta)$ and sends it to $\mathcal{A}$. In the previous hybrid, an honest $\mathsf{P_A}$ computes $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}] = \mathsf{K}[\boldsymbol{e}_u]\cdot\mathbf{A}_1+\mathsf{K}[\boldsymbol{r}]\cdot\mathbf{A}_2+\mathsf{K}[\boldsymbol{e}_r]+\mathsf{com}_{\boldsymbol{e}_u}\cdot\Delta$, which holds the property that $\mathsf{M}[\boldsymbol{e}_u] = \mathsf{K}[\boldsymbol{e}_u]+\boldsymbol{e}_u\cdot\Delta$, $\mathsf{M}[\boldsymbol{e}_r] = \mathsf{K}[\boldsymbol{e}_r]+\boldsymbol{e}_r\cdot\Delta$, and $\mathsf{M}[\boldsymbol{r}] = \mathsf{K}[\boldsymbol{r}]+\boldsymbol{r}\cdot\Delta$. Consequently, the $\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}]$ computed by simulator equals the one computed by honest $\mathsf{P_A}$ in real world. Thus, this hybrid is identical to the previous one.

This concludes the proof.

□

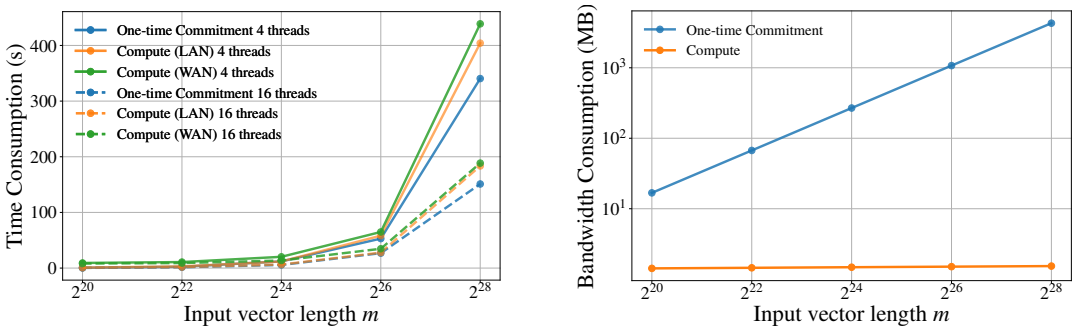
A.3 Performance of Primal-LPN based C-VOLE

Complexity Analysis. The primal-LPN-based C-VOLE protocol requires MPVOLE instantiations of $\boldsymbol{e}_u \in \mathbb{F}^{w\cdot 2^L}$ and $\boldsymbol{e}_r \in \mathbb{F}^{t\cdot 2^{L'}}$, VOLE instantiation of $\boldsymbol{r} \in \mathbb{F}^k$, together with a $H(\mathsf{M}[\mathsf{com}_{\boldsymbol{e}_u}])$ for consistency check and a commitment with a size of $N+m$, where $N = t\cdot 2^{L'}$. According to the instantiation of MPVOLE protocol in Section A.5, the sparse vector of length $w\cdot 2^L$ requires communication overhead of $wL\kappa$. For instantiation of VOLE protocol in Section A.9, it is dominated by $k\log|\mathbb{F}|$ for input vector with length k. Thus, the communication complexity is $(wL+tL')\kappa + (k+N+m)\log|\mathbb{F}|$ in total.

Parameter Selection of Hybrid Commitment. Same as the dual-LPN-based hybrid commitment parameter selection, we use EA code to choose the (w, L) pair of $\boldsymbol{e}_u \in \mathbb{F}^{w\cdot 2^L}$ for $\boldsymbol{x} \in \mathbb{F}^m$ with 128-bit security level. Given the (w, L) pair, we fix the structure of $\boldsymbol{e}_u$ and select parameters to construct a 128-bit secure commitment $\mathsf{com}_{\boldsymbol{e}_u}$, where $\mathsf{com}_{\boldsymbol{e}_u} = \boldsymbol{e}_u\cdot\mathbf{A}_1+\boldsymbol{r}\cdot\mathbf{A}_2+\boldsymbol{e}_r$. According to Theorem 1, we select the (k, t, L') parameters for the primal LPN commitment. For each $\boldsymbol{x} \in \mathbb{F}^m$ of different lengths, groups of parameters (w, L, k, t, L') are computed. Based on the complexity analysis above, we minimize the communication cost for primal-LPN-based C-VOLE protocol and present the commitment parameters that achieve 128 bit statistically binding and computational hiding for $\boldsymbol{x} \in \{\mathbb{F}^{2^{20}}, \mathbb{F}^{2^{24}}, \mathbb{F}^{2^{28}}\}$ in Table 8.

Performance Evaluation. The performance of primal-LPN-based C-VOLE also consists of two parts: computing the one-time commitment of sender's input vector $\boldsymbol{x}$ and a online computation with any individual receiver. The time consumption and communication overhead of these two phases are shown in Figure 9a and Figure 9b respectively.

For one-time commitment, the sender generates a primal-LPN-based hybrid commitment of input vector $\boldsymbol{x} \in \mathbb{F}^m$ and publishes it to all other parties. The computation cost for the sender is 0.59 s for the computation of a vector of length $m = 2^{20}$ and 340.46 s for $m = 2^{28}$ using 4 threads,



(a) Time consumption.

(b) Bandwidth consumption.

Figure 9: **Performance of the C-VOLE protocol based on the primal-LPN assumption in LAN and WAN settings.**

Input	Intermediate sparse vector			Dual LPN commitment			
$\|\boldsymbol{x}\|$	$\|\boldsymbol{e}_u\|$	w	L	$\|\boldsymbol{e}_r\|$	t	L'	$\|\mathsf{com}_{\boldsymbol{e}_u}\|(n)$
2^{20}	$5 * 2^{20}$	640	13	1640	165	4	882
2^{24}	$5 * 2^{24}$	640	17	2688	168	4	885
2^{28}	$5 * 2^{28}$	640	21	2736	171	4	888

Table 9: Dual-LPN-based hybrid commitment parameters selection of $\boldsymbol{x}$ with different lengths. Using aggressive parameters for EA-code. Commitment achieves 128-bit security level.

while it takes 0.30 s for $m = 2^{20}$ and 151.06 s for $m = 2^{28}$ using 16 threads. The communication overhead of publishing the primal-LPN-based hybrid commitment is 16.80 MB for $m = 2^{20}$ and 4295.0 MB for $m = 2^{28}$. For $m = 2^{20}$, the compute phase takes 0.83 s under a LAN network and 9.19 s under a WAN network with a bandwidth of 50 Mbps when using 4 threads. With 16 threads, the compute phase takes 0.61 seconds under LAN network and 8.13 seconds under WAN network. For $m = 2^{28}$, the compute phase with 4 threads takes 404.11 s under a LAN network and 435.81 s under a WAN network. With 16 threads, the corresponding times are 183.47 s under a LAN network and 188.56 s under a WAN network. Regarding the online communication overhead, the cost is 1.73 Mb for $m = 2^{20}$ and 1.79 MB for $m = 2^{28}$.

Comparison with Dual-LPN-based C-VOLE. As we can see from the performance evaluation of dual-LPN-based C-VOLE and primal-LPN-based C-VOLE, both schemes perform well with the chosen hybrid commitment parameters. Considering the cumulative effect of both communication and computation overhead, the scheme based on primal-LPN shows slightly better concrete performance than the one based on dual-LPN in our experiments.

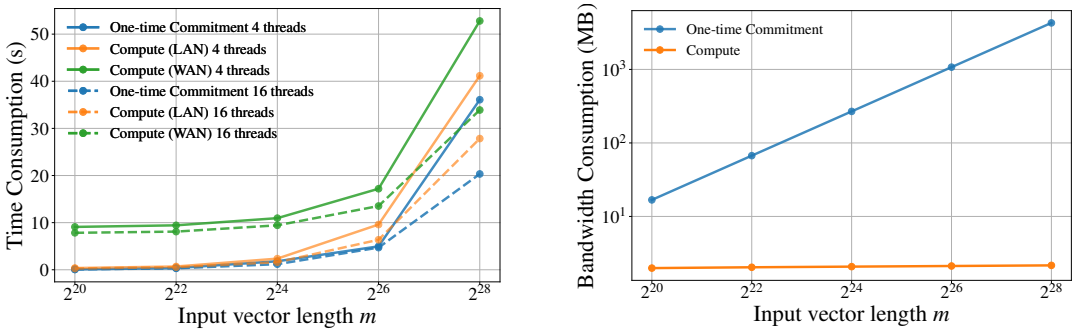
A.4 Performance Evaluation of C-VOLE with Aggressive EA-code

According to Section A.1.2, we also evaluate the performance of our C-VOLE protocol over field $\mathbb{F}_2^{128}$ using an aggressive EA-code, which remains secure against attacks [RRT23] applicable only to $\mathbb{F}_2$. We took $\ell = 9$ and compute the hamming weight $t = 640$. Taking dual-LPN commitment as an example, Table 9 presents the parameters for commitments that achieve 128 bit security level.

Performance Evaluation. The performance of dual-LPN-based C-VOLE using aggressive EA-code also shown in two parts: computing the one-time commitment of sender's input vector $\boldsymbol{x}$ and a online computation with any individual receiver. The time consumption and communication

overhead of these two phases are shown in Figure 10a and Figure 10b respectively.

For one-time commitment, the computation cost for the sender is 0.07 s for the computation of a vector of length $m = 2^{20}$ and 36.08 s for $m = 2^{28}$ using 4 threads, while it takes 0.07 s for $m = 2^{20}$ and 20.33 s for $m = 2^{28}$ using 16 threads. The communication overhead of publishing the commitment is 16.79 MB for $m = 2^{20}$ and 4294.98 MB for $m = 2^{28}$. For $m = 2^{20}$, the compute phase takes 0.36 s under a LAN network and 9.09 s under a WAN network with a bandwidth of 50 Mbps when using 4 threads. With 16 threads, the compute phase takes 0.26 seconds under LAN network and 7.84 seconds under WAN network. For $m = 2^{28}$, the compute phase with 4 threads takes 41.16 s under a LAN network and 52.79 s under a WAN network. With 16 threads, the corresponding times are 27.84 s under a LAN network and 33.89 s under a WAN network. Regarding the online communication overhead, the cost is 1.99 Mb for $m = 2^{20}$ and 2.16 MB for $m = 2^{28}$.



(a) Time consumption.

(b) Bandwidth consumption.

Figure 10: **Performance of the C-VOLE protocol based on the dual-LPN assumption using aggressive EA-code in LAN and WAN settings.**

Comparison with Dual-LPN-based C-VOLE with conservative parameters. As we can see from the performance evaluation, the aggressive parameters result in significantly more efficient time consumption, which is roughly 10 times better for $m = 2^{28}$, although this comes at the cost of slightly worse online communication.

A.5 Construction of MPVOLE

Here, we briefly introduce the construction of protocol Π_{MPVOLE} in Figure 11, which securely instantiates the $\mathcal{F}_{\mathsf{MPVOLE}}$ functionality in Figure 3. This protocol is a slightly variation of the single-point subfield-VOLE protocol from [WYKW21], where the consistency check is slightly adjust to suit the VOLE protocol. A brief description of the protocol is as follows:

The Π_{MPVOLE} generates VOLE relation of sparse multi-point vector $\boldsymbol{e}_u \in \mathbb{F}^{t \cdot 2^h}, n = 2^h$, which satisfies a specific form that it is a concatenation of t single-point vectors $\boldsymbol{e}_u^i, i \in [t]$, each with length n. We let α_i to represent the index of non-zero element that $\alpha_i \in [n]$ and β_i to denote the value of the non-zero element in block $\boldsymbol{e}_u^i, i \in [t]$. This protocol first utilize $\mathcal{F}_{\mathsf{VOLE}}$ over field $\mathbb{F}$ to generate VOLE of $\beta_i, i \in [t]$, as selected by $\mathsf{P_A}$. Then, for each single-point block $\boldsymbol{e}_u^i, i \in [t]$, this protocol follows the design of [YWL+20]. It first let $\mathsf{P_B}$ run GGM algorithm [GGM84], which is an algorithm building pseudorandom function by using a PRG, detailed in Section A.7, to obtain pseudorandom output $\boldsymbol{v}_i$, $\mathsf{P_A}$ run GGM' algorithm, also detailed in Section A.7, to obtain $\boldsymbol{w}_i, i \neq \alpha_i$. Then $\mathsf{P_B}$ sends d_i to help $\mathsf{P_A}$ recover $\boldsymbol{w}_i$. For consistency check, this protocol first generates a random value $r_0 \leftarrow \mathbb{F}$ and its VOLE relation $\mathsf{M}[r_0] = \mathsf{K}[r_0] + r_0 \cdot \Delta$. With sampled $\chi_i[j], i \in [t], j \in [n]$,

$\mathsf{P_A}$ computes linear combination of $w_i[j], i \in [t], j \in [n]$ and $\mathsf{P_B}$ computes linear combination of $v_i[j], i \in [t], j \in [n]$. $\mathsf{P_A}$ uses r_0 to mask the sum of $\beta_i \cdot \chi_i[\alpha_i], i \in [t]$ and sends the masked value to $\mathsf{P_B}$. Then, the Equation 3 holds if both parties acts honestly.

$$\begin{aligned} &\sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j]\boldsymbol{w}_i[j] - \mathsf{M}[r_0] \\ =&\sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j]\boldsymbol{v}_i[j] - \mathsf{K}[r_0] + (\sum_{i\in[t]}\beta_i\chi_i[\alpha_i] - r_0)\Delta \end{aligned} \tag{3}$$

We present the detailed protocol in Figure 11. We also prove the protocol securely instantiate $\mathcal{F}_{\mathsf{MPVOLE}}$ in $(\mathcal{F}_{\mathsf{RVOLE}}, \mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{OT}})$ hybrid model against malicious adversary in Theorem 6. Moreover, the construction of Π_{VOLE} is introduced in Section A.9.

A.6 Security Analysis of MPVOLE

Theorem 6. *Protocol Π_{MPVOLE} in Figure 11 securely instantiated $\mathcal{F}_{\mathsf{MPVOLE}}$ in Figure 3 in ($\mathcal{F}_{\mathsf{RVOLE}}$, $\mathcal{F}_{\mathsf{VOLE}}$, $\mathcal{F}_{\mathsf{OT}}$)-hybrid model against malicious adversaries.*

Proof. Let $\mathcal{A}$ be a PPT adversary that corrupt $\mathsf{P_A}$ or $\mathsf{P_B}$. We construct a PPT simulator $\mathcal{S}$ with access to $\mathcal{F}_{\mathsf{MPVOLE}}$ and simulates the adversary's view. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the output of $\mathcal{S}$ and the honest party in the ideal world.

Corrupted $\mathsf{P_A}$. Let $\mathcal{S}$ access $\mathcal{F}_{\mathsf{MPVOLE}}$ as an honest $\mathsf{P_A}$ and interact with $\mathcal{A}$ as an honest $\mathsf{P_B}$. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

For Initialize phase, $\mathcal{S}$ simulates as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{RVOLE}}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ sends (init) to $\mathcal{F}_{\mathsf{MPVOLE}}$.

For Extend phase, $\mathcal{S}$ simulates as follows:

2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{RVOLE}}$ and receives $(\mathsf{extend}, 1)$ from $\mathcal{A}$. $\mathcal{S}$ receives $r_0 \in \mathbb{F}$, $\mathsf{M}[r_0] \in \mathbb{F}$ from $\mathcal{A}$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ and receives $(\mathsf{extend}, \beta, \mathsf{t})$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{M}[\beta] \in \mathbb{F}^t$ from $\mathcal{A}$.
4. For each iteration $i \in [t]$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{OT}}$ and receives $\bar{a}_i[k], k \in [h]$. $\mathcal{S}$ reconstructs $\alpha_i, i \in [t]$. $\mathcal{S}$ samples $K^k_{i,\bar{a}_i[k]} \leftarrow \mathbb{F}, k \in [h]$ and sends them to $\mathcal{A}$. $\mathcal{S}$ reconstructs $\boldsymbol{e}_u$ and sends $(\mathsf{extend}, \boldsymbol{e}_u, h, t)$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{S}$ also runs $\{v_i[j]\}_{j\neq\alpha_i} := \mathsf{GGM'}(\alpha_i, \{K^k_{i,\bar{a}_i[k]}\}_{k\in[h]})$.
5. For each iteration $i \in [t]$, $\mathcal{S}$ samples $d_i \leftarrow \mathbb{F}$ and sends it to $\mathcal{A}$. $\mathcal{S}$ computes $\boldsymbol{w}_i[j] := v_i[j]$ for $j \neq \alpha_i$ and $\boldsymbol{w}_i[\alpha_i] := \mathsf{M}[\beta_i] - \left(d_i + \sum_{j\neq\alpha_i}\boldsymbol{w}_i[j]\right)$.
6. $\mathcal{S}$ receives $\left(\{\chi_i[j]\}_{i\in[t],j\in[n]}, x^*\right)$ from $\mathcal{A}$. $\mathcal{S}$ checks whether $x^* = \sum_{i\in[t]}\beta_i \cdot \chi_i[\alpha_i] - r_0$. If it is, $\mathcal{S}$ computes $V_B = \sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j] \cdot \boldsymbol{w}_i[j] - \mathsf{M}[r_0]$ and sends $H(V_B)$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ sends $H(V_B) \leftarrow \mathbb{F}$ to $\mathcal{A}$.
7. $\mathcal{S}$ constructs $\mathsf{M}[\boldsymbol{e}_u] = \boldsymbol{w}_1||\cdots||\boldsymbol{w}_t$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $(\mathcal{F}_{\mathsf{RVOLE}}, \mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{OT}})$ model.

Protocol Π_{MPVOLE}

Parameters: GGM: $(\boldsymbol{v}, \{S_{i,0}, S_{i,1}\}_{i\in[\log N]}) \leftarrow GGM(N, s), s \leftarrow \{0,1\}^\kappa, \boldsymbol{v} \in \mathbb{F}^N, S_{i,0} \in \{0,1\}^\kappa, S_{i,1} \in \{0,1\}^\kappa$; GGM': $\{w_h\}_{h\in[N]/\{\alpha\}} \leftarrow GGM'(\alpha, \{S_{i,\bar{\alpha_i}}\}_{i\in[\log N]}), w_h \in \{0,1\}^\kappa$. Both algorithms are detailed in Section A.7.

Inputs: For each extend sparse execution, $\mathsf{P_A}$ holds regular sparse multi-point vector $\boldsymbol{e}_u \in \mathbb{F}^{t\cdot 2^h}, n = 2^h$. $\mathsf{P_A}$ partition $\boldsymbol{e}_u \in \mathbb{F}^{t\cdot 2^h}$ into t blocks, that for each block $\boldsymbol{e}_u^i \in \mathbb{F}^n, i \in [t]$, it has one non-zero element $\beta_i \in \mathbb{F}$ at position $\alpha_i \in [n]$. $\mathsf{P_A}$ represents α_i as bit vectors, namely $\boldsymbol{a}_i \in \{0,1\}^h$. Both $\mathsf{P_A}$ and $\mathsf{P_B}$ have integers $t \in \mathbb{N}$ and $n = 2^h$ for some $h \in \mathbb{N}$.

Initialize:

1. Both $\mathsf{P_A}$ and $\mathsf{P_B}$ sends (init) to $\mathcal{F}_{\mathsf{RVOLE}}$. $\mathsf{P_B}$ receives $\Delta \in \mathbb{F}$ from $\mathcal{F}_{\mathsf{RVOLE}}$.

Extend:

2. Both $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{RVOLE}}$ with $(\mathsf{extend}, 1)$. The functionality sends $r_0 \in \mathbb{F}, \mathsf{M}[r_0] \in \mathbb{F}$ to $\mathsf{P_A}$, and $\mathsf{K}[r_0] \in \mathbb{F}$ to $\mathsf{P_B}$, such that $\mathsf{M}[r_0] = \mathsf{K}[r_0] + r_0 \cdot \Delta$.
3. $\mathsf{P_A}$ call $\mathcal{F}_{\mathsf{VOLE}}$ with $(\mathsf{extend}, \beta, \mathsf{t})$, where β is a concatenation of β_is, and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{VOLE}}$ with $(\mathsf{extend}, \mathsf{t})$. The functionality sends $\mathsf{M}[\beta] \in \mathbb{F}^t$ to $\mathsf{P_A}$, and $\mathsf{K}[\beta] \in \mathbb{F}^t$ to $\mathsf{P_B}$, such that $\mathsf{M}[\beta_i] = \mathsf{K}[\beta_i] + \beta_i\Delta$ for each $i \in [t]$.
4. For each iteration $i \in [t]$, $\mathsf{P_B}$ samples seed $l_i \leftarrow \{0,1\}^\kappa$, runs $\mathsf{GGM}(1^n, l_i)$ to obtain $\left(\{v_i[j]\}_{j\in[n]}, \{(K^k_{i,0}, K^k_{i,1})\}_{k\in[h]}\right)$, and sets $\boldsymbol{v}_i[j] := v_i[j]$ for $j \in [n]$. For $k \in [h]$, $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{OT}}$ with input $(K^k_{i,0}, K^k_{i,1})$; $\mathsf{P_A}$ call $\mathcal{F}_{\mathsf{OT}}$ with input $\bar{a}_i[k]$ and get $K^k_{i,\bar{a}_i[k]}$. Then $\mathsf{P_A}$ runs $\{v_i[j]\}_{j\neq\alpha_i} := \mathsf{GGM'}(\alpha_i, \{K^k_{i,\bar{a}_i[k]}\}_{k\in[h]})$.
5. For each iteration $i \in [t]$, $\mathsf{P_B}$ sends $d_i := \mathsf{K}[\beta_i] - \sum_{j\in[n]} \boldsymbol{v}_i[j] \in \mathbb{F}$ to $\mathsf{P_A}$. Then, $\mathsf{P_A}$ defines $\boldsymbol{w}_i \in \mathbb{F}^n$ as the vector with $\boldsymbol{w}_i[j] := v_i[j]$ for $j \neq \alpha_i$ and $\boldsymbol{w}_i[\alpha_i] := \mathsf{M}[\beta_i] - \left(d_i + \sum_{j\neq\alpha_i} \boldsymbol{w}_i[j]\right)$. Note that $\boldsymbol{w}_i = \boldsymbol{v}_i + \Delta \cdot \boldsymbol{e}_u^i$.
6. $\mathsf{P_A}$ and $\mathsf{P_B}$ check **GGM-consistency** as follows.
   (a) $\mathsf{P_A}$ samples $\chi_i[j] \leftarrow \mathbb{F}$ for $i \in [t], j \in [n]$. $\mathsf{P_A}$ then computes $x^* := \sum_{i\in[t]} \beta_i \cdot \chi_i[\alpha_i] - r_0 \in \mathbb{F}$ and sends $\left(\{\chi_i[j]\}_{i\in[t],j\in[n]}, x^*\right)$ to $\mathsf{P_B}$, who computes $y := \mathsf{K}[r_0] - \Delta \cdot x^*$.
   (b) $\mathsf{P_A}$ computes $V_A := \sum_{i\in[t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \boldsymbol{w}_i[j] - \mathsf{M}[r_0] \in \mathbb{F}$, while $\mathsf{P_B}$ computes $V_B := \sum_{i\in[t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \boldsymbol{v}_i[j] - y \in \mathbb{F}$. Then $\mathsf{P_B}$ sends $H(V_B)$ to $\mathsf{P_A}$. $\mathsf{P_A}$ aborts if $H(V_A) \neq H(V_B)$.
7. $\mathsf{P_A}$ outputs $\mathsf{M}[\boldsymbol{e}_u] = \boldsymbol{w}_1 || \cdots || \boldsymbol{w}_t \in \mathbb{F}^{tn}$ and $\mathsf{P_B}$ outputs $\mathsf{K}[\boldsymbol{e}_u] = \boldsymbol{v}_1 || \cdots || \boldsymbol{v}_t \in \mathbb{F}^{tn}$.

Figure 11: Multi-point VOLE protocol.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{RVOLE}}$, $\mathcal{F}_{\mathsf{VOLE}}$, $\mathcal{F}_{\mathsf{OT}}$ and simulate messages to $\mathcal{A}$.

For step (4), $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{OT}}$ and sends $K^k_{i,\bar{a}_i[k]} \leftarrow \mathbb{F}, k \in [h]$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, an honest $\mathsf{P_B}$ computes $K^k_{i,\bar{a}_i[k]}, k \in [h]$ according to GGM algorithm with random seed $l_i \leftarrow \{0,1\}^\kappa$. By leveraging PRG over random seed l_i, $K^k_{i,\bar{a}_i[k]}, k \in [h]$ are indistinguishable from random value over $\mathbb{F}$.

For step (5), $\mathcal{S}$ samples $d_i \leftarrow \mathbb{F}, i \in [t]$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, an honest $\mathsf{P_B}$ computes $d_i = \mathsf{K}[\beta_i] - \sum_{j\in[n]} \boldsymbol{v}_i[j]$ to $\mathcal{A}$. Since $\mathsf{K}[\beta_i]$ is unknown to $\mathcal{A}$, d_i is uniform distributed to $\mathcal{A}$.

For step (6), $\mathcal{S}$ receives $\left(\{\chi_i[j]\}_{i\in[t],j\in[n]}, x^*\right)$ from $\mathcal{A}$ and sends $H(V_B)$ to $\mathcal{A}$. $\mathcal{S}$ sets $H(V_B) =$

$H(\sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j]\cdot\boldsymbol{w}_i[j]-\mathsf{M}[r_0])$ if $x^*=\sum_{i\in[t]}\beta_i\cdot\chi_i[\alpha_i]-r_0$. $\mathcal{S}$ sets $H(V_B)\leftarrow\mathbb{F}$ if $x^*\neq\sum_{i\in[t]}\beta_i\cdot\chi_i[\alpha_i]-r_0$. In Hybrid $\mathcal{H}_0$, an honest $\mathsf{P_B}$ computes $V_B=\sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j]\cdot\boldsymbol{v}_i[j]-y$, where $y=\mathsf{K}[r_0]-\Delta\cdot\ x^*$, and sends $H(V_B)$ to $\mathcal{A}$. If $x^*=\sum_{i\in[t]}\beta_i\cdot\chi_i[\alpha_i]-r_0+rand, rand\in\mathbb{F}$, $y=\mathsf{M}[r_0]-\sum_{i\in[t]}\beta_i\cdot\chi_i[\alpha_i]\cdot\Delta-rand\cdot\Delta$. We have the following equation holds.

$$
\begin{aligned}
V_B &= \sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j]\cdot\boldsymbol{v}_i[j]-y\\
&= \sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j]\boldsymbol{v}_i[j]-\mathsf{M}[r_0]+\sum_{i\in[t]}\beta_i\chi_i[\alpha_i]\Delta+rand\Delta\\
&= \sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j]\cdot\boldsymbol{w}_i[j]-\sum_{i\in[t]}\beta_i\cdot\chi_i[\alpha_i]\cdot\Delta-\mathsf{M}[r_0]\\
&\quad+\sum_{i\in[t]}\beta_i\cdot\chi_i[\alpha_i]\cdot\Delta+rand\cdot\Delta\\
&= \sum_{i\in[t]}\sum_{j=0}^{n-1}\chi_i[j]\cdot\boldsymbol{w}_i[j]-\mathsf{M}[r_0]+rand\cdot\Delta\\
&= V_A+rand\cdot\Delta
\end{aligned}
$$

If $rand=0$, we have $V_A=V_B$. Thus, the simulation is identical to the real world protocol execution. Otherwise, $\mathcal{A}$ will receives $H(V_B)=H(V_A+rand\cdot\Delta)$, which is indistinguishable from a random value in the hybrid of random oracle model.

Thus, this hybrid is identical to the previous one.

Corrupted $\mathsf{P_B}$. Let $\mathcal{S}$ access $\mathcal{F}_{\mathsf{MPVOLE}}$ as an honest $\mathsf{P_B}$ and interact with $\mathcal{A}$ as an honest $\mathsf{P_A}$. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$. For $\mathsf{Initialize}$ phase, $\mathcal{S}$ simulates as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{RVOLE}}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ receives Δ from $\mathcal{A}$.

For Extend phase, $\mathcal{S}$ simulates as follows:

2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{RVOLE}}$ and receives $(\mathsf{extend},1)$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{K}[r_0]\leftarrow\mathbb{F}$ from $\mathcal{A}$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ and receives $(\mathsf{extend},\mathsf{t})$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathsf{K}[\beta]\leftarrow\mathbb{F}^t$ from $\mathcal{A}$. $\mathcal{S}$ samples $\beta_i, i\in[t]$ and computes $\mathsf{M}[\beta_i]=\mathsf{K}[\beta_i]+\beta_i\cdot\Delta, i\in[t]$.
4. For each iteration $i\in[t]$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{OT}}$ and receives $(K^k_{i,0},K^k_{i,1}), k\in[h]$.
5. For each iteration $i\in[t]$, $\mathcal{S}$ receives $d_i\leftarrow\mathbb{F}$ from $\mathcal{A}$. For each $\alpha_i\in[n]$, $\mathcal{S}$ computes $\boldsymbol{w}_i^{\alpha_i}$ as follows:
   (a) Computes $\{v_i^{\alpha_i}[j]\}_{j\neq\alpha_i}:=\mathsf{GGM'}(\alpha_i,\{K^k_{i,\bar{a}_i[k]}\}_{k\in[h]})$ and sets $\boldsymbol{w}_i^{\alpha_i}[j]=v_i^{\alpha_i}[j]$ for $j\neq\alpha_i$
   (b) Sets $\boldsymbol{w}_i^{\alpha_i}[\alpha_i]:=\mathsf{M}[\beta_i]-\Big(d_i+\sum_{j\neq\alpha_i}\boldsymbol{w}_i^{\alpha_i}[j]\Big)$.
6. (a) S samples $\big(\{\chi_i[j]\leftarrow\mathbb{F}\}_{i\in[t],j\in[n]}, x^*\leftarrow\mathbb{F}\big)$ to $\mathcal{A}$.

(b) $\mathcal{S}$ receives $H(V_B)$ from $\mathcal{A}$. $\mathcal{S}$ is able to extract V_B as $\mathcal{S}$ emulate random oracle H for $\mathcal{A}$.

For each $\{\alpha_i \in [n]\}_{i\in[t]}$, denoted as $\boldsymbol{\alpha} \in \mathcal{N}^t, \mathcal{N} := [n]$, $\mathcal{S}$ computes $V_A^{\boldsymbol{\alpha}}$ as follows:$V_A^{\boldsymbol{\alpha}} = \sum_{i\in[t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \boldsymbol{w}_i^{\alpha_i}[j] - \mathsf{M}[r_0]$. Define $I := \{\boldsymbol{\alpha} \in \mathcal{N}^t | V_A^{\boldsymbol{\alpha}} = V_B\}$. We also define $I = I_1 \times \cdots \times I_t$, where $I_i := \{\alpha_i \in \mathcal{N} | V_A^{\boldsymbol{\alpha}} = V_B\}$. $\mathcal{S}$ aborts if $I = \varnothing$.

$\mathcal{S}$ chooses an arbitrary $\boldsymbol{\alpha} \in I$ and computes vector $\boldsymbol{v}_i$ for each $i \in [t]$ as follows:

(a) Sets $\boldsymbol{v}_i[j] := \boldsymbol{w}_i^{\alpha_i}[j]$ for $j \neq \alpha_i$

(b) Sets $\boldsymbol{v}_i[\alpha_i] := \mathsf{K}[\beta_i] - d_i - \sum_{j\neq\alpha_i} \boldsymbol{v}_i[j]$

$\mathcal{S}$ send $\{I_1, \cdots, I_t\}$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. If it receives abort, $\mathcal{S}$ aborts.

7. $\mathcal{S}$ constructs $\mathsf{K}[\boldsymbol{e}_u] = \boldsymbol{v}_1 || \cdots || \boldsymbol{v}_t$ to $\mathcal{F}_{\mathsf{MPVOLE}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $(\mathcal{F}_{\mathsf{RVOLE}}, \mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{OT}})$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{RVOLE}}$, $\mathcal{F}_{\mathsf{VOLE}}$, $\mathcal{F}_{\mathsf{OT}}$ and simulate messages to $\mathcal{A}$.

For step (6), $\mathcal{S}$ samples $\left(\{\chi_i[j] \leftarrow \mathbb{F}\}_{i\in[t],j\in[n]}, x^* \leftarrow \mathbb{F}\right)$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, an honest $\mathsf{P_A}$ samples $\{\chi_i[j] \leftarrow \mathbb{F}\}_{i\in[t],j\in[n]}$, and computes $x^* = \sum_{i\in[t]} \beta_i \cdot \chi_i[\alpha_i] - r_0$. Since r_0 is output by $\mathcal{F}_{\mathsf{VOLE}}$ to $\mathsf{P_A}$ and is uniformly distributed against $\mathcal{A}$, x^* is uniformly distributed against $\mathcal{A}$, which is indistinguishable from this hybrid. Thus, this hybrid is identical to the previous one.

Hybrid $\mathcal{H}_2$ Same as Hybrid $\mathcal{H}_1$ except $\mathcal{S}$ aborts at step (6) if $I = \varnothing$ or received abort from $\mathcal{F}_{\mathsf{MPVOLE}}$.

For step (6), $\mathcal{S}$ extracts set I and send it to $\mathcal{F}_{\mathsf{MPVOLE}}$, which is equivalent to $\mathcal{A}$ samples selective failure attack on $\mathsf{P_A}$'s chosen non-zero index $\alpha_i, i \in [t]$. If $\mathcal{S}$ receives abort from $\mathcal{F}_{\mathsf{MPVOLE}}$, there exist $i, i \in [t]$, $\alpha_i \notin I_i$, that is equivalent to $\boldsymbol{\alpha} \notin I$. In the real world protocol execution, if $V_B \neq V_A^{\boldsymbol{\alpha}}$, $\mathsf{P_A}$ aborts. This is equivalent to $\boldsymbol{\alpha} \notin I$. Thus, $\mathcal{F}_{\mathsf{MPVOLE}}$ aborts if and only if the real world protocol execution aborts. This hybrid is identical to the previous one.

Then, we will prove the outputs of $\mathcal{A}$ and honest party in the real world is indistinguishable from the outputs of $\mathcal{S}$ and honest party in the ideal world.

We will prove that except with probability $1/|\mathbb{F}|$, all choices of $\alpha_i \in I_i$ in the above simulation lead to the same vector $\boldsymbol{v}_i$.

Since $V_A^{\boldsymbol{\alpha}} = V_A^{\boldsymbol{\alpha}'} = V_B$, we have:

$$V_A^{\boldsymbol{\alpha}} = V_A^{\boldsymbol{\alpha}'}$$

$$\sum_{i\in[t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \boldsymbol{w}_i^{\alpha_i}[j] - \mathsf{M}[r_0] = \sum_{i\in[t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \boldsymbol{w}_i^{\alpha_i'}[j] - \mathsf{M}[r_0]$$

$$\begin{aligned}
\sum_{i\in[t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \boldsymbol{w}_i^{\alpha_i}[j] - y - \sum_{i\in[t]} \beta_i \chi_i[\alpha_i]\Delta = &\sum_{i\in[t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \boldsymbol{w}_i^{\alpha_i'}[j] - y - \sum_{i\in[t]} \beta_i \chi_i[\alpha_i']\Delta \\
&\sum_{i\in[t]} \Big(\sum_{j\neq\alpha_i,\alpha_i'} \chi_i[j](\boldsymbol{w}_i^{\alpha_i}[j] - \boldsymbol{w}_i^{\alpha_i'}[j]) + \chi_i[\alpha_i](\boldsymbol{w}_i^{\alpha_i}[\alpha_i] \\
&- \boldsymbol{w}_i^{\alpha_i'}[\alpha_i] - \beta_i\Delta) + \chi_i[\alpha_i'](\boldsymbol{w}_i^{\alpha_i}[\alpha_i'] - \boldsymbol{w}_i^{\alpha_i'}[\alpha_i'] + \beta_i\Delta) \Big) \\
&= 0
\end{aligned}$$

Since $\{\chi_i[j]\}_{i\in[t],j\in[n]}$ are uniformly distributed and independent from $\boldsymbol{w}_i^{\alpha_i}$, $\boldsymbol{w}_i^{\alpha_i'}$, and Δ, we have except with probability $1/|\mathbb{F}|$,

$$\boldsymbol{w}_i^{\alpha_i}[j] = \boldsymbol{w}_i^{\alpha_i'}[j], j \neq \alpha_i, \alpha_i' \tag{4}$$

$$\boldsymbol{w}_i^{\alpha_i}[\alpha_i] - \boldsymbol{w}_i^{\alpha_i'}[\alpha_i] = \boldsymbol{w}_i^{\alpha_i'}[\alpha_i'] - \boldsymbol{w}_i^{\alpha_i}[\alpha_i'] = \Delta\beta_i \tag{5}$$

Thus, we are able to obtain for any $\alpha_i, \alpha_i' \in I_i$, $\boldsymbol{v}_i^{\alpha_i}[j] = \boldsymbol{v}_i^{\alpha_i'}[j], j \neq \{\alpha_i, \alpha_i'\}$ from Equation 4. According to $\boldsymbol{w}_i^{\alpha_i}[\alpha_i] = \boldsymbol{v}_i^{\alpha_i}[\alpha_i] + \Delta\beta_i$ and Equation 5, we are able to get $\boldsymbol{v}_i^{\alpha_i}[\alpha_i] = \boldsymbol{w}_i^{\alpha_i'}[\alpha_i] = \boldsymbol{v}_i^{\alpha_i'}[\alpha_i]$ and similarly $\boldsymbol{v}_i^{\alpha_i'}[\alpha_i'] = \boldsymbol{w}_i^{\alpha_i}[\alpha_i'] = \boldsymbol{v}_i^{\alpha_i}[\alpha_i']$. □

A.7 GGM Algorithm

We detailed the GGM algorithm [BGI15, BGI16, BCG+17] and its evaluation in this section to help with the construction of Π_{MPVOLE}.

- $\mathsf{GGM}(N, s)$: On input $N = 2^d$ for some $d \in \mathbb{N}$ and $s \in \{0,1\}^\kappa$, this algorithm executes as follows:
  1. Define $s_0^0 := s$. For $i \in [1, d)$ and $j \in [0, 2^{i-1})$ compute $\left(s_{2j}^i, s_{2j+1}^i\right) := G(s_j^{i-1})$, where $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ is a PRG.
  2. Compute the leaves as $\left(s_{2j}^d, s_{2j+1}^d\right) := G'(s_j^{d-1})$ for $j \in [0, 2^{d-1})$, where $G' : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ is another PRG. Define a vector $\boldsymbol{v}$ as $v_h := s_{h-1}^d$ for $h \in [N]$.
  3. For $i \in [d]$, compute $S_{i,0} := \sum_{j\in[0,2^{i-1})} s_{2j}^i$ and $S_{i,1} := \sum_{j\in[0,2^{i-1})} s_{2j+1}^i$, where for $i \in [1, d)$, the addition is defined over $\mathbb{F}_{2^\kappa}$; for $i = d$, the addition is defined over $\mathbb{F}$.
  4. Output $\left(\boldsymbol{v}, \{(S_{i,0}, S_{i,1})\}_{i\in[d]}\right)$.
- $\mathsf{GGM}'(\alpha, \{S_{i,\bar{\alpha}_i}\}_{i\in[d]})$: On input $\alpha \in [N]$ and a set $\{S_{i,\bar{\alpha}_i}\}_{i\in[d]}$ where $\bar{\alpha}_i = \alpha_i \oplus 1 \in \{0,1\}$ for $i \in [d]$ and $\alpha = \sum_{i=1}^d \alpha_i \cdot 2^{i-1} + 1 \in [N]$, for $i \in [d]$, this algorithm does the following:
  1. Define an i-bit string $\alpha_i^* := \alpha_1 \cdots \alpha_{i-1}\bar{\alpha}_i$.
  2. If $i = 1$, define $s_{\bar{\alpha}_1}^1 := S_{1,\bar{\alpha}_1}$.
  3. If $i \geq 2$, for $j \in [0, 2^{i-1}), j \neq \alpha_1 \cdots \alpha_{i-1}$, compute $(s_{2j}^i, s_{2j+1}^i) := G(s_j^{i-1})$ if $i < d$ and $(s_{2j}^d, s_{2j+1}^d) := G'(s_j^{d-1})$ otherwise.
  4. Compute $s_{\alpha_i^*}^i := S_{i,\bar{\alpha}_i} + \left(\sum_{j\in[0,2^{i-1}),j\neq\alpha_1\cdots\alpha_{i-1}} s_{2j+\bar{\alpha}_i}^i\right)$, where the addition is defined over $\mathbb{F}_{2^\kappa}$ if $i \in [1, d)$ and over $\mathbb{F}$ otherwise.
  5. Define $w_h := s_{h-1}^d$ for $h \in [N], h \neq \alpha$, and output $\{w_h\}_{h\in[N]\setminus\{\alpha\}}$.

A.8 Random VOLE

Random Vector Oblivious Linear Evaluation. Random Vector Oblivious Linear Evaluation (RVOLE) functionality is a two party functionality that samples a vector $\boldsymbol{x} \leftarrow \mathbb{F}^m$ to $\mathsf{P_A}$, a field element $\Delta \leftarrow \mathbb{F}$ and a vector $\mathsf{K}[\boldsymbol{x}] \leftarrow \mathbb{F}^m$ to $\mathsf{P_B}$, and delivers $\mathsf{M}[\boldsymbol{x}] = \mathsf{K}[\boldsymbol{x}] + \boldsymbol{x}\cdot\Delta \in \mathbb{F}^m$ to $\mathsf{P_A}$. RVOLE can be efficiently generated by LPN-based approaches [BCGI18, WYKW21, YWL+20, SGRR19]. This paper takes the functionality of random VOLE depicted in Figure 12 as the fundamental building block to construct committed VOLE.

Functionality $\mathcal{F}_{\mathsf{RVOLE}}$

Initialize: Upon receiving (init) from both $\mathsf{P_A}$ and $\mathsf{P_B}$, sample $\Delta \leftarrow \mathbb{F}$ and send it to $\mathsf{P_B}$. The functionality stores Δ. If $\mathsf{P_B}$ is malicious, receive $\Delta \in \mathbb{F}$ from $\mathsf{P_B}$.

Extend: Upon receiving (extend, m) from both $\mathsf{P_A}$ and $\mathsf{P_B}$:

1. Samples $\mathsf{K}[\boldsymbol{x}] \leftarrow \mathbb{F}^m$. If $\mathsf{P_B}$ is malicious, receive $\mathsf{K}[\boldsymbol{x}] \in \mathbb{F}^m$ from adversary.
2. Samples $\boldsymbol{x} \leftarrow \mathbb{F}^m$ and computes $\mathsf{M}[\boldsymbol{x}] := \mathsf{K}[\boldsymbol{x}] + \Delta \cdot \boldsymbol{x} \in \mathbb{F}^m$. If $\mathsf{P_A}$ is malicious, receives $(\boldsymbol{x} \in \mathbb{F}^m, \mathsf{M}[\boldsymbol{x}] \in \mathbb{F}^m)$ from the adversary, and recomputes $\mathsf{K}[\boldsymbol{x}] := \mathsf{M}[\boldsymbol{x}] - \Delta \cdot \boldsymbol{x} \in \mathbb{F}^m$
3. Sends $(\boldsymbol{x}, \mathsf{M}[\boldsymbol{x}])$ to $\mathsf{P_A}$ and $(\mathsf{K}[\boldsymbol{x}])$ to $\mathsf{P_B}$.

Figure 12: The random VOLE functionality.

A.9 Protocol of VOLE

We present a simple VOLE protocol constructed from $\mathcal{F}_{\mathsf{RVOLE}}$. In this protocol, $\mathsf{P_A}$ holds the designated input vector $\boldsymbol{b} \in \mathbb{F}^m$. After calling $\mathcal{F}_{\mathsf{RVOLE}}$ on both sides, $\mathsf{P_A}$ obtains random $(\boldsymbol{r}, \mathsf{M}[\boldsymbol{r}]) \in \mathbb{F}^{2m}$ and $\mathsf{P_B}$ obtains $(\Delta, \mathsf{K}[\boldsymbol{r}]) \in \mathbb{F}^{m+1}$, such that $\mathsf{M}[\boldsymbol{r}] = \mathsf{K}[\boldsymbol{r}] + \boldsymbol{r} \cdot \Delta$. Then, $\mathsf{P_A}$ sends the difference $\boldsymbol{f}$ between $\boldsymbol{r}$ and $\boldsymbol{b}$. Consequently, both parties are able to obtain the VOLE correlation of the designated vector $\boldsymbol{b}$, where $\mathsf{M}[\boldsymbol{b}] = \mathsf{M}[\boldsymbol{r}]$ and $\mathsf{K}[\boldsymbol{b}] = \mathsf{K}[\boldsymbol{r}] + \boldsymbol{f} \cdot \Delta$.

This protocol is able to securely instantiate $\mathcal{F}_{\mathsf{VOLE}}$ in the $\mathcal{F}_{\mathsf{RVOLE}}$ hybrid model. A simple proof sketch is provided here. For corrupted $\mathsf{P_A}$, a simulator $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ and receives $\boldsymbol{r} \in \mathbb{F}^m$, $\mathsf{M}[\boldsymbol{r}] \in \mathbb{F}^m$ from $\mathcal{A}$. $\mathcal{S}$ receives $\boldsymbol{f} \in \mathbb{F}^m$ from $\mathcal{A}$. It is able to recover $\boldsymbol{b} = \boldsymbol{r} - \boldsymbol{f}$, and $\mathsf{M}[\boldsymbol{b}] = \mathsf{M}[\boldsymbol{r}]$, and sends $\mathsf{M}[\boldsymbol{b}]$ to $\mathcal{F}_{\mathsf{VOLE}}$. The simulator does not need to simulate any message to the adversary. For corrupted $\mathsf{P_B}$, $\mathcal{S}$ receives $\mathsf{K}[\boldsymbol{r}] \in \mathbb{F}^m$ from $\mathcal{A}$ and samples $\boldsymbol{f} \leftarrow \mathbb{F}^m$ to $\mathcal{A}$, which is indistinguishable from the $\boldsymbol{f}$ in real protocol. Then, $\mathcal{S}$ is able to compute $\mathsf{K}[\boldsymbol{b}] = \mathsf{K}[\boldsymbol{r}] + \boldsymbol{f}\Delta$ to $\mathcal{F}_{\mathsf{VOLE}}$. The above simulation could simulate the view of adversary in real protocol. Also, $\mathcal{S}$ output what $\mathcal{A}$ outputs could ensure the same output distribution of the real protocol.

A.10 Proof of Theorem 4

Proof. Let $\mathcal{A}$ be a PPT adversary that corrupt P_1 or P_j. We construct a PPT simulator $\mathcal{S}$ with access to $\mathcal{F}_{\mathsf{PSI}}$ and simulates the adversary's view. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the output of $\mathcal{S}$ and the honest party in the ideal world.

Corrupted P_1. Let $\mathcal{S}$ access $\mathcal{F}_{\mathsf{PSI}}$ as an honest P_1 and interact with $\mathcal{A}$ as an honest P_2. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

(1) $\mathcal{S}$ emulates random oracle H_1 and receives elements $x_i, i \in [n]$ from $\mathcal{A}$. $\mathcal{S}$ samples $H_1(x_i) \leftarrow \mathbb{F}, i \in [n]$ and sends them to $\mathcal{A}$.

(2) $\mathcal{S}$ receives $\mathsf{Hcom}_{\boldsymbol{p}} \in \mathbb{F}^N$ from $\mathcal{A}$.

(3) $\mathcal{S}$ samples $c^s \leftarrow \mathbb{F}$ and sends it to $\mathcal{A}$.

(4) $\mathcal{S}$ receives $w^r \in \mathbb{F}, r \in \{0,1\}^\kappa$ from $\mathcal{A}$.

(5) $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{C-VOLE}}$ and receives $(\mathsf{init}, \boldsymbol{p}, \mathsf{Hdecom}_{\boldsymbol{p}})$, where $\boldsymbol{p} \in \mathbb{F}^m$, $\mathsf{Hdecom}_{\boldsymbol{p}} \in \mathbb{F}^{m'}$ from $\mathcal{A}$. $\mathcal{S}$ checks whether $(\mathsf{Hcom}_{\boldsymbol{p}}, \mathsf{Hdecom}_{\boldsymbol{p}})$ opens to $\boldsymbol{p}$. If not, $\mathcal{S}$ sends abort to $\mathcal{A}$. Otherwise, $\mathcal{S}$ waits

to receive $\mathsf{M}[\boldsymbol{p}] \leftarrow \mathbb{F}^m$ from $\mathcal{A}$. $\mathcal{S}$ checks whether $\mathsf{Decode}(\boldsymbol{p}, x_i, r) = H(x_i)$. If it is, $\mathcal{S}$ inserts x_i to set X. $\mathcal{S}$ sends (init, X) to $\mathcal{F}_{\mathsf{PSI}}$ and waits to receive (setid, Z).

(6) $\mathcal{S}$ samples $w^s \leftarrow \mathbb{F}$ and programs random oracle H_1 that $c^s = H_1(w^s)$. $\mathcal{S}$ sends w^s to $\mathcal{A}$. $\mathcal{S}$ computes $w = w^s + w^r$.

(7) $\mathcal{S}$ samples $ey_i \leftarrow \{0,1\}^{out}$ for each $y_i \in Y$ and inserts ey_i to set EY. $\mathcal{S}$ sends EY to $\mathcal{A}$.

(8) $\mathcal{S}$ emulates random oracle H_2 and receives query (q_i, x_i) from $\mathcal{A}$. $\mathcal{S}$ first checks whether the query is fresh.

(a) If it is, $\mathcal{S}$ checks whether $x_i \in Z$ and $q_i = \mathsf{Decode}(\mathsf{M}[\boldsymbol{p}], x_i, r) + w$. If it is, $\mathcal{S}$ samples ex_i from EY/EX and sends it to $\mathcal{A}$. $\mathcal{S}$ records the triple (q_i, x_i, ex_i) and inserts ex_i to EX. If not, S samples $ex_i \leftarrow \{0,1\}^{out}/(EY \cup EX)$ to $\mathcal{A}$, records the triple, and inserts ex_i to EX.

(b) If not, $\mathcal{S}$ checks records and sends corresponding ex_i to $\mathcal{A}$.

$\mathcal{S}$ aborts if $\mathcal{A}$ aborts and outputs what $\mathcal{A}$ outputs.

While $\mathcal{S}$ access $\mathcal{F}_{\mathsf{PSI}}$ as an honest P_1 and interact with $\mathcal{A}$ as an honest $P_j, j \in [3, n]$. $\mathcal{S}$ simulates the **Extend** phase. The simulation is same as above except step (5):

(5) $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{C-VOLE}}$ and receives $(\mathsf{init}, \boldsymbol{p}, \mathsf{Hdecom}_{\boldsymbol{p}})$, where $\boldsymbol{p} \in \mathbb{F}^m, \mathsf{Hdecom}_{\boldsymbol{p}} \in \mathbb{F}^{m'}$ from $\mathcal{A}$. $\mathcal{S}$ checks whether ($\mathsf{Hcom}_{\boldsymbol{p}}$, $\mathsf{Hdecom}_{\boldsymbol{p}}$) opens to $\boldsymbol{p}$. If not, $\mathcal{S}$ sends abort to $\mathcal{A}$. Otherwise, $\mathcal{S}$ waits to receive $\mathsf{M}[\boldsymbol{p}] \leftarrow \mathbb{F}^m$ from $\mathcal{A}$. $\mathcal{S}$ sends $(\mathsf{init}, \mathsf{setid})$ to $\mathcal{F}_{\mathsf{PSI}}$ and waits to receive Z.

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $(\mathcal{F}_{\mathsf{C-VOLE}}, H_1, H_2)$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{C-VOLE}}$, random oracle H_1, H_2, and simulates the messages to $\mathcal{A}$ as follows:

For step (1), $\mathcal{S}$ emulates H_1 and samples $H_1(x_i) \leftarrow \mathbb{F}, i \in [n]$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, $H_1(x_i) \leftarrow \mathbb{F}, i \in [n]$ are generated and outputted by a random oracle, which is uniform distributed and indistinguishable from this hybrid.

For step (3), $\mathcal{S}$ samples $c^s \leftarrow \mathbb{F}$ to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, Random oracle H_1 outputs c^s, which is uniformly distributed over $\mathbb{F}$ and indistinguishable from this hybrid.

For step (6), $\mathcal{S}$ samples $w^s \leftarrow \mathbb{F}$ to $\mathcal{A}$. $\mathcal{S}$ also programs random oracle H_1 that $c^s = H_1(w_s)$. In hybrid $\mathcal{H}_0$, an honest P_2 samples $w^s \leftarrow \mathbb{F}$ and computes $c^s = H_1(w_s)$. The distribution of w^s is indistinguishable from this hybrid.

For step (7), $\mathcal{S}$ programs EY to size $|Y|$ with random elements in $\{0,1\}^{out}$. In hybrid H_0, an honest P_2 computes ey_i from random oracle for each element $y_i \in Y_2$. Thus, EY is indistinguishable from this hybrid.

For step (8), $\mathcal{S}$ emulates H_2 and programs H_2 that for $x_i \in Z, q_i = \mathsf{Decode}(\mathsf{M}[\boldsymbol{p}], x_i, r) + w$, it sends $ex_i \in EY$ to $\mathcal{A}$. Thus, the output of $\mathcal{A}$ is same as interacting with an honest P_2.

Thus, this hybrid is identical to the previous one.

Corrupted P_j**.** Let $\mathcal{S}$ access $\mathcal{F}_{\mathsf{PSI}}$ as an honest P_j and interact with $\mathcal{A}$ as an honest P_1. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

(1-2) $\mathcal{S}$ samples $\mathsf{Hcom}_{\boldsymbol{p}} \leftarrow \mathbb{F}^N$. $\mathcal{S}$ publishes $\mathsf{Hcom}_{\boldsymbol{p}}$.

(3) $\mathcal{S}$ emulates H_1 and receives w^s from $\mathcal{A}$. $\mathcal{S}$ samples $c^s \leftarrow \mathbb{F}$ to $\mathcal{A}$. $\mathcal{S}$ receives $c'^s \leftarrow \mathbb{F}$ from $\mathcal{A}$. If $c'^s \neq c^s$, $\mathcal{S}$ aborts in step (6).

(4) $\mathcal{S}$ samples $w^r \leftarrow \mathbb{F}, r \leftarrow \{0,1\}^\kappa$ and sends them to $\mathcal{A}$.

(5) $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{C-VOLE}}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ waits to receive $(\Delta, \mathsf{K}[\boldsymbol{p}]) \leftarrow \mathbb{F}^{m+1}$ from $\mathcal{A}$.

(6) $\mathcal{S}$ receives w'^s from $\mathcal{A}$ and checks whether $w'^s = w^s$. If not, $\mathcal{S}$ aborts.

(7) $\mathcal{S}$ emulates H_1, H_2, and receives query as follows:

(a) $\mathcal{S}$ receives y_i from $\mathcal{A}$. If the query is fresh, $\mathcal{S}$ samples $H_1(y_i) \leftarrow \mathbb{F}$ to $\mathcal{A}$ and stores the pair $(y_i, H_1(y_i))$. If not, $\mathcal{S}$ checks the records and sends corresponding $H_1(y_i)$ to $\mathcal{A}$.

(b) $\mathcal{S}$ receives query (q_i, y_i) from $\mathcal{A}$. If the query is fresh, $\mathcal{S}$ samples $ey_i \leftarrow \{0,1\}^{out}$ to $\mathcal{A}$ and records triple (y_i, q_i, ey_i). If not, $\mathcal{S}$ checks the records and sends corresponding ey_i to $\mathcal{A}$.

$\mathcal{S}$ receives EY from $\mathcal{A}$. For each $ey_i \in EY$, $\mathcal{S}$ checks whether it is recorded and $q_i = \mathsf{Decode}(\mathsf{K}[\boldsymbol{p}], y_i, r) + \Delta H_1(y_i) + w$. If it is, $\mathcal{S}$ inserts y_i to set Y and sends $(\mathsf{intersect}, Y)$ to $\mathcal{F}_{\mathsf{PSI}}$. $\mathcal{S}$ receives setid from $\mathcal{F}_{\mathsf{PSI}}$.

While $\mathcal{S}$ access $\mathcal{F}_{\mathsf{PSI}}$ as an honest P_j and interact with $\mathcal{A}$ as an honest P_1. $\mathcal{S}$ simulates the **Extend** phase. The simulation is same as above except: $\mathcal{S}$ extracts set Y and sends $(\mathsf{intersect}, Y, \mathsf{setid})$ to $\mathcal{F}_{\mathsf{PSI}}$ at step (7).

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $(\mathcal{F}_{\mathsf{C-VOLE}}, H_1, H_2)$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{C-VOLE}}$, random oracle H_1, H_2, and simulates the messages to $\mathcal{A}$ as follows:

For step (1-2), $\mathcal{S}$ samples $\mathsf{Hcom}_{\boldsymbol{p}} \leftarrow \mathbb{F}^N$, and publishes it. In Hybrid $\mathcal{H}_0$, $\mathsf{Hcom}_{\boldsymbol{p}}$ generated from commitment scheme. According to Theorem 2, $\mathsf{Hcom}_{\boldsymbol{p}}$ is computational indistinguishable from pseudorandom vector over $\mathbb{F}^N$, which is indistinguishable from this hybrid.

For step (3), $\mathcal{S}$ emulates H_1 and samples $c^s \leftarrow \mathbb{F}$ for each w^s to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, c^s is uniformly distributed over $\mathbb{F}$ and indistinguishable from this hybrid.

For step (4), $\mathcal{S}$ samples $w^r \leftarrow \mathbb{F}, r \leftarrow \{0,1\}^k$ as an honest P_1 does in hybrid $\mathcal{H}_0$.

For step (7), $\mathcal{S}$ emulates H_1, H_2, and samples $H_1(y_i) \leftarrow \mathbb{F}, ey_i \leftarrow \{0,1\}^{out}$ to $\mathcal{A}$, which is indistinguishable from the output of a random oracle. $\mathcal{S}$ records all the programmed pairs and triple. $\mathcal{S}$ receives EY from $\mathcal{A}$ and checks each $ey_i \in EY$ whether been recorded and the previous query is computed correctly to extract $\mathcal{A}$'s private set elements and sends them to $\mathcal{F}_{\mathsf{PSI}}$. Thus, the output of $\mathcal{S}$ and honest party is same as the output of $\mathcal{A}$ and honest party.

Thus, this hybrid is identical to the previous one.

Hybrid $\mathcal{H}_2$ Same as Hybrid $\mathcal{H}_1$ except $\mathcal{S}$ aborts at step (6) if $c'^s \neq c^s$ or $w'^s \neq w^s$. It is indistinguishable from P_1 aborts if $c^s \neq H_1(w^s)$ in hybrid $\mathcal{H}_1$. Thus, this hybrid is identical to the previous one.

This concludes the proof. □