# Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits

Chenkai Weng
Northwestern University
ckweng@u.northwestern.edu

Kang Yang
State Key Laboratory of Cryptology
yangk@sklc.org

Jonathan Katz*
George Mason University
jkatz2@gmail.com

Xiao Wang
Northwestern University
wangxiao@cs.northwestern.edu

**Abstract**

Efficient zero-knowledge (ZK) proofs for arbitrary boolean or arithmetic circuits have recently attracted much attention. Existing solutions suffer from either significant prover overhead (superlinear running time and/or high memory usage) or relatively high communication complexity (at least $\kappa$ bits per gate, for computational security parameter $\kappa$ and boolean circuits). We show here a new protocol for constant-round interactive ZK proofs that simultaneously allows for a highly efficient prover and low communication. Specifically:

- The prover in our protocol has linear running time and, perhaps more importantly, memory usage linear in the memory needed to evaluate the circuit non-cryptographically. This allows our proof system to scale easily to very large circuits.
- For circuits of size $C$ over an arbitrary finite field and a statistical security parameter $\rho$, the communication complexity of our protocol is roughly $3B+1$ elements per gate, where $B=1$ for large fields and $B=\frac{\rho}{\log C}$ for small fields.

Using 5 threads and a 50 Mbps network, our ZK protocol ($\rho=40, \kappa=128$) runs at a rate of $0.54\mu s$/gate for a boolean circuit with 10 billion gates, using only 400 MB of memory and communicating 9 bits/gate. This is roughly an order of magnitude faster than prior work.

## 1 Introduction

Zero-knowledge (ZK) proofs (of knowledge) [GMR85, GMW86] are fundamental cryptographic tools. They allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$, who holds a circuit $\mathcal{C}$, that the prover knows a witness $w$ for which $\mathcal{C}(w)=1$ without leaking any extra information. While ZK proofs for arbitrary circuits are possible [GMW86], historically such proofs were inefficient as they relied on reductions to generic NP-complete problems. Over the past decade, however, several ZK proof systems have been developed that yield far more efficient protocols. These include zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) [Gro10, GGPR13, BCG+13, BCTV14, BCC+16, BBB+18, WTs+18, BCR+19, BBHR19], ZK proofs based on interactive oracle proofs (IOPs) and techniques from the setting of verifiable outsourcing [GKR08, ZGK+17, XZZ+19, ZXZS20], ZK proofs following the "MPC-in-the-head" approach [IKOS07, GMO16, CDG+17, AHIV17, KKW18, dDOS19], and a line of work constructing efficient ZK proofs from garbled circuits (ZKGC) [JKO13,

*Work done as a consultant for Stealth Software Technologies, Inc.

| Protocol Type | libSNARK [BCTV14] zk-SNARK | Virgo [ZXZS20] IOP-based | Ligero [AHIV17] MPC-in-the-head | [HK20] ZKGC | This work |
|---|---|---|---|---|---|
| Prover time | 360 s | 53 s | 400 s | 49 s | 5.7 s |
| Verifier time | 0.002 s | 0.05 s | 4 s | 49 s | 5.7 s |
| Communication | 0.13 KB | 253 KB | 1.5 MB | 182.2 MB | 12.4 MB |
| Memory Usage | $\gg$ 10 GB | $\geq$ 10 GB | $\geq$ 10 GB | $\leq$ 500 MB | $\leq$ 500 MB |

Table 1: **Computing a ZK proof of knowledge of all leaves in a depth-9 Merkle tree based on SHA-256 (256 leaves; 511 hash-function evaluations).** Performance of our protocol (with $\rho = 40$, $\kappa = 128$) is measured by running the prover and verifier on two machines, each using 5 threads connected via a 50 Mbps network. Numbers for other protocols are taken from the indicated references or directly from [XZZ+19], and use an experimental setup no slower than ours. The memory usage numbers are either conservative estimation from prior reports, or private communication with the authors.

FNO15, HK20]. Each of these works offers different tradeoffs between underlying assumptions (both computational hardness assumptions as well as setup assumptions), round complexity (in particular, whether the proof requires interaction or can be made non-interactive), expressiveness (e.g., whether the scheme handles boolean or arithmetic circuits), and efficiency. With regard to efficiency, measures of interest include the prover complexity, the verifier complexity, and the total communication as a function of the circuit size.

Focusing on efficiency, existing work (see Table 1) can be characterized roughly as either having very short proofs (e.g., sublinear in the circuit size, or in some cases even sublinear in the length of a witness) but poor prover complexity[1], as in the case of zk-SNARKs and ZK schemes following the IOP-based approach, or achieving good prover complexity but having high communication overhead, as in the case of MPC-in-the-head and ZKGC schemes.

In this work we propose a new approach to ZK proofs that enables an extremely efficient prover while having lower communication complexity than comparable approaches. As in the ZKGC approach, we obtain prover complexity linear in the circuit size. Moreover, the prover can process the circuit "on-the-fly," meaning that the prover's memory usage is linear in the memory needed to evaluate the circuit non-cryptographically; this allows our protocol to scale easily to very large circuits. At the same time, we achieve communication complexity that is more than an order of magnitude lower than what can be achieved using the ZKGC approach. We compare our work to prior work in Table 1.

The main drawback of our protocol—shared by the ZKGC approach—is that it requires interaction; we do not know how to use our techniques to obtain a non-interactive proof even in the random-oracle model. Our proof system does, however, have a non-interactive *online* phase following an interactive offline phase that can be executed by the parties before the circuit is known.

## 1.1 Outline of Our Solution

Our ZK protocol can be separated into two phases: an interactive offline phase that can be executed by the prover and verifier before both the circuit and the witness are known, and an online phase that can be made non-interactive in the random-oracle model. We describe the online phase first, since we view this as our main conceptual contribution.

[1]Here we refer not only to the prover's running time, but also its memory usage. Schemes that impose significant memory requirements do not scale well to very large circuits.

**Online phase.** The online phase of our protocol can be viewed as adapting the core idea of the ZKGC approach by viewing a ZK proof as a special case of secure two-party computation (2PC) where one party has no input. We differ from the ZKGC approach in the underlying 2PC protocol we use as our starting point: rather than using garbled circuits, we instead rely on a "GMW-style" approach [GMW87] using authenticated multiplication triples [Bea92, NNOB12, DPSZ12] (whose values are known to the prover) generated during the offline phase. A drawback of GMW-style protocols in the context of secure computation is that they have round complexity linear in the depth of the circuit being evaluated. Crucially, in the ZK context, we can exploit the fact that only one party has an input to obtain a protocol that runs in constant rounds (or even one round in the random-oracle model).

The prover and verifier run in linear time since they each make only one pass over the circuit. Moreover, they can evaluate the circuit "on-the-fly" (i.e., with memory overhead linear in what is needed to evaluate the circuit non-cryptographically), which allows our protocol to scale easily to handle very large circuits. Our approach is communication-efficient as well: for a circuit with $C$ multiplication gates over an arbitrary finite field $\mathbb{F}_p$, the marginal communication is only about $3B + 1$ elements per multiplication gate, where $B = 1$ if $\log p > \rho$ and $B = \frac{\rho}{\log C}$ otherwise for a statistical security parameter $\rho$.

**Instantiating the offline phase.** To give a ZK proof for a circuit with $C$ multiplication gates over an arbitrary field $\mathbb{F}_p$, we use subfield Vector Oblivious Linear Evaluation (sVOLE) [BCGI18] to set up $BC$ authenticated triples in the offline phase. For boolean circuits (i.e., $p = 2$), we use the recent work by Kang et al. [YWL+20] to generate an initial pool of authenticated bits, and then use these authenticated bits to generate authenticated triples as in prior work [NO09]. For $p > 2$, we extend their protocol to support larger fields, and obtain an sVOLE protocol for any finite field, which may be of independent interest. We defer the details to Section 4.

**Comparison to prior work [BCGI18, BCG+19b].** Boyle et al. [BCGI18, BCG+19b] also proposed ZK proofs in which VOLE is used during an offline phase to set up the correlated randomness between the prover and the verifier, and the online phase is non-interactive. Focusing on the online phase, the primary advantages of their work are that it does not require the random-oracle model, and it can be run multiple times following a single execution of the offline phase (that is, the offline phase is *reusable*). The focus of our work is concrete efficiency, which was not investigated by Boyle et al. Moreover, our work applies to circuits over arbitrary fields, whereas the work of Boyle et al. applies either to circuits over large fields [BCGI18] or boolean circuits [BCG+19b]. We also offer concrete efficiency improvements for the offline phase; some of our improvements could be applied to the protocols of Boyle et al. as well. We defer further discussion to Section 4, and provide a detailed performance evaluation in Section 5.

**Overview of the paper.** After reviewing some preliminaries in Section 2, we describe the online phase of our ZK proof in Section 3. In Section 4 we describe the details of the offline phase. We conclude with our experimental evaluation in Section 5.

## 2 Preliminaries

We use $\kappa$ and $\rho$ to denote the computational and statistical security parameters, respectively. We let $\mathsf{negl}(\cdot)$ denote some unspecified negligible function, and use log to denote logarithms in base 2. We write $x \leftarrow S$ to denote sampling $x$ uniformly from a finite set $S$, and $x \leftarrow \mathcal{D}$ to denote sampling $x$ according to a distribution $\mathcal{D}$. We set $[a, b] = \{a, \ldots, b\}$ and $[a, b) = \{a, \ldots, b-1\}$, and write $[n]$ as shorthand for $[1, n]$. We use bold lower-case letters like $\boldsymbol{a}$ for row vectors, and bold upper-case

letters like $\mathbf{A}$ for matrices. We let $\boldsymbol{a}[i]$ denote the $i$th component of a vector $\boldsymbol{a}$ (with $\boldsymbol{a}[0]$ the first entry), and let $\boldsymbol{a}[i:j]$ and $\boldsymbol{a}[i:j)$ represent the subvectors $(\boldsymbol{a}[i],\ldots,\boldsymbol{a}[j])$ and $(\boldsymbol{a}[i],\ldots,\boldsymbol{a}[j-1])$, respectively.

A circuit $\mathcal{C}$ over a field $\mathbb{F}_p$ is defined by a set of input wires $\mathcal{I}_{\mathsf{in}}$ and output wires $\mathcal{I}_{\mathsf{out}}$, along with a list of gates of the form $(\alpha,\beta,\gamma,T)$, where $\alpha,\beta$ are the indices of the input wires of the gate, $\gamma$ is the index of the output wire of the gate, and $T \in \{\mathsf{Add},\mathsf{Mult}\}$ is the type of the gate. If $p=2$, then $\mathcal{C}$ is a boolean circuit with $\mathsf{Add}=\oplus$ and $\mathsf{Mult}=\wedge$. If $p>2$ is a prime, then $\mathcal{C}$ is an arithmetic circuit where $\mathsf{Add}/\mathsf{Mult}$ correspond to addition/multiplication in $\mathbb{F}_p$. We let $C$ denote the number of $\mathsf{Mult}$ gates in the circuit.

When we work in an extension field $\mathbb{F}_{p^r}$ of $\mathbb{F}_p$, we fix some monic, irreducible polynomial $f(X)$ of degree $r$ and so $\mathbb{F}_{p^r} \cong \mathbb{F}_p[X]/f(X)$. We let $g \in \mathbb{F}_{p^r}$ denote the element corresponding to $X \in \mathbb{F}_p[X]/f(X)$; thus, every element $w \in \mathbb{F}_{p^r}$ can be written uniquely as $w = \sum_{i=0}^{r-1} w_i \cdot g^i$ with $w_i \in \mathbb{F}_p$, and we may view elements of $\mathbb{F}_{p^r}$ equivalently as vectors in $\mathbb{F}_p^r$. When we write arithmetic expressions involving both elements of $\mathbb{F}_p$ and elements of $\mathbb{F}_{p^r}$, it is understood that values in $\mathbb{F}_p$ are viewed as lying in $\mathbb{F}_{p^r}$ in the natural way. We let $\mathbb{F}^*$ denote the nonzero elements of a field $\mathbb{F}$.

### 2.1 Information-Theoretic MACs and Batch Opening

We use *information-theoretic message authentication codes* (IT-MACs) [NNOB12, DPSZ12] to authenticate values in a finite field $\mathbb{F}_p$, where authentication is done in an extension field $\mathbb{F}_{p^r} \supseteq \mathbb{F}_p$. In more detail, let $\Delta \in \mathbb{F}_{p^r}$ be a *global key*, sampled uniformly, that is known only by one party $\mathsf{P_B}$. A value $x \in \mathbb{F}_p$ known by the other party $\mathsf{P_A}$ can be authenticated by giving $\mathsf{P_B}$ a uniform key $\mathsf{K}[x] \in \mathbb{F}_{p^r}$ and giving $\mathsf{P_A}$ the corresponding MAC tag

$$\mathsf{M}[x] = \mathsf{K}[x] + \Delta \cdot x \in \mathbb{F}_{p^r}.$$

We denote such an authenticated value by $[x]$. Note that authenticated values are additively homomorphic, i.e., if $\mathsf{P_A}$ and $\mathsf{P_B}$ hold authenticated values $[x],[x']$ then they can locally compute $[x''] = [x+x']$ by having $\mathsf{P_A}$ set $x'' := x + x'$ and $\mathsf{M}[x''] := \mathsf{M}[x] + \mathsf{M}[x']$ and having $\mathsf{P_B}$ set $\mathsf{K}[x''] := \mathsf{K}[x] + \mathsf{K}[x']$. Similarly, for a public value $b \in \mathbb{F}_p$, the parties can locally compute $[y] = [x+b]$. We denote these operations by $[x''] = [x] + [x']$ and $[y] = [x] + b$, respectively.

We extend the above notation to vectors of authenticated values as well. In that case, $[\boldsymbol{u}]$ means that (for some $n$) $\mathsf{P_A}$ holds $\boldsymbol{u} \in \mathbb{F}_p^n$ and $\boldsymbol{w} \in \mathbb{F}_{p^r}^n$, while $\mathsf{P_B}$ holds $\boldsymbol{v} \in \mathbb{F}_{p^r}^n$ with $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$.

An *authenticated multiplication triple* consists of authenticated values $[x],[y],[z]$ where $z = x \cdot y$.

**Batch opening of authenticated values.** An authenticated value $[x]$ can be "opened" by having $\mathsf{P_A}$ send $x \in \mathbb{F}_p$ and $\mathsf{M}[x] \in \mathbb{F}_{p^r}$ to $\mathsf{P_B}$, who then verifies that $\mathsf{M}[x] \stackrel{?}{=} \mathsf{K}[x] + \Delta \cdot x$. This has soundness error $1/p^r$, and requires sending an additional $r \log p$ bits (beyond $x$ itself). While it is possible to repeat this in parallel when opening multiple authenticated values $[x_1],\ldots,[x_\ell]$, the communication complexity can be reduced using batching [NNOB12, DPSZ12, NST17]. We describe two such methods here.

One approach relies on a cryptographic hash function $\mathsf{H}$. Specifically, $\mathsf{P_A}$ sends (in addition to the values $x_1,\ldots,x_\ell$ themselves) a digest $h := \mathsf{H}(\mathsf{M}[x_1],\ldots,\mathsf{M}[x_\ell])$ of all the tags; $\mathsf{P_B}$ then checks that $h \stackrel{?}{=} \mathsf{H}(\mathsf{K}[x_1]+\Delta\cdot x_1,\ldots,\mathsf{K}[x_\ell]+\Delta\cdot x_\ell)$. Modeling $\mathsf{H}$ as a random oracle with $2\kappa$-bit output, it is not hard to see that the soundness error (i.e., the probability that $\mathsf{P_A}$ can successfully lie about *any* value) is at most $(q_{\mathsf{H}}^2+1)/2^{2\kappa} + 1/p^r$, where $q_{\mathsf{H}}$ denotes the number of queries $\mathsf{P_A}$ makes to $\mathsf{H}$. The communication overhead is only $2\kappa$ bits, independent of $\ell$.

A second approach, which is entirely information theoretic, works as follows:

1. $\mathsf{P_A}$ sends $x_1, \ldots, x_\ell \in \mathbb{F}_p$ to $\mathsf{P_B}$.

2. $\mathsf{P_B}$ picks uniform $\chi_1, \ldots, \chi_\ell \in \mathbb{F}_{p^r}$ and sends them to $\mathsf{P_A}$.

3. $\mathsf{P_A}$ computes $\mathsf{M}[x] := \sum_{i=1}^{\ell} \chi_i \cdot \mathsf{M}[x_i] \in \mathbb{F}_{p^r}$, and sends $\mathsf{M}[x]$ to $\mathsf{P_B}$.

4. $\mathsf{P_B}$ computes $x := \sum_{i=1}^{\ell} \chi_i \cdot x_i \in \mathbb{F}_{p^r}$ and $\mathsf{K}[x] := \sum_{i=1}^{\ell} \chi_i \cdot \mathsf{K}[x_i] \in \mathbb{F}_{p^r}$. It accepts the opened values iff $\mathsf{M}[x] \stackrel{?}{=} \mathsf{K}[x] + \Delta \cdot x$.

The soundness error of this approach is given by the following lemma.

**Lemma 1.** *Fix $x_1, \ldots, x_\ell \in \mathbb{F}_p$ and $\mathsf{M}[x_1], \ldots, \mathsf{M}[x_\ell] \in \mathbb{F}_{p^r}$ known to $\mathsf{P_A}$, and say a uniform $\Delta \in \mathbb{F}_{p^r}$ and $\{\mathsf{K}[x_i] = \mathsf{M}[x_i] - \Delta \cdot x_i\}$ are given to $\mathsf{P_B}$. For uniform $\chi_1, \ldots, \chi_\ell \in \mathbb{F}_{p^r}$, the probability that $\mathsf{P_A}$ can successfully open values $(x'_1, \ldots, x'_\ell) \neq (x_1, \ldots, x_\ell)$ to $\mathsf{P_B}$ is at most $2/p^r$.*

*Proof.* Fix $(x'_1, \ldots, x'_\ell) \neq (x_1, \ldots, x_\ell)$ sent by $\mathsf{P_A}$ in the first step. Set $\omega \stackrel{\text{def}}{=} \sum_{i=1}^{\ell} \chi_i \cdot (x'_i - x_i)$. By a standard argument, $\Pr[\omega = 0] \leq 1/p^r$.

Assume $\omega \neq 0$. If $\mathsf{P_A}$ sends $\mathsf{M}$, then $\mathsf{P_B}$ accepts only if

$$\begin{aligned} \mathsf{M} &= \sum_{i=1}^{\ell} \chi_i \cdot \mathsf{K}[x_i] + \Delta \cdot \sum_{i=1}^{\ell} \cdot \chi_i \cdot x'_i \\ &= \sum_{i=1}^{\ell} \chi_i \cdot (\mathsf{M}[x_i] - \Delta \cdot x_i) + \Delta \cdot \sum_{i=1}^{\ell} \chi_i \cdot x'_i \\ &= \sum_{i=1}^{\ell} \chi_i \cdot \mathsf{M}[x_i] + \Delta \cdot \omega. \end{aligned}$$

Everything in the final expression is fixed and known to $\mathsf{P_A}$ except for $\Delta$. Moreover, $\mathsf{P_A}$ succeeds iff $\Delta = \omega^{-1} \cdot (\mathsf{M} - \sum_{i-1}^{\ell} \chi_i \cdot \mathsf{M}[x_i])$, which occurs with probability $1/p^r$. The lemma follows. □

We can make this second approach non-interactive, using the Fiat-Shamir heuristic in the random-oracle model, by computing the $\{\chi_i\}$ as a hash $\mathsf{H}$ of the $\{x_i\}$ values sent by $\mathsf{P_A}$ in the first step. Adapting the above proof, one can show that this has soundness error at most $(q_{\mathsf{H}} + 1)/p^r$.

Hereafter, we write $\mathsf{Open}([\boldsymbol{x}])$ to denote a generic batch opening of a vector of authenticated values. In addition, we write $\mathsf{CheckZero}([\boldsymbol{x}])$ for the special case where all $x_i$ are supposed to be 0 and so need not be sent. We let $\varepsilon_{\mathsf{open}}$ denote the soundness error (which depends on the technique used); note that when using either of the techniques described above, this probability is independent of the number $\ell$ of authenticated values opened.

## 2.2 Security Model and Functionalities

We use the framework of universal composability (UC) [Can01] to prove security in the presence of a malicious, static adversary. We say that a protocol $\Pi$ *UC-realizes* an ideal functionality $\mathcal{F}$ if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, there is a PPT simulator $\mathcal{S}$, such that for any PPT environment $\mathcal{Z}$ with arbitrary auxiliary input $z$, the distribution of the output of $\mathcal{Z}$ in the *ideal-world* execution where the parties interact with $\mathcal{F}$ is computationally indistinguishable from the output of $\mathcal{Z}$ in the *real-world* execution where the parties execute $\Pi$.

The protocol we construct in this work realizes the (standard) zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}$, reproduced in Figure 1 for convenience. (We omit session identifiers in all our ideal functionalities

**Functionality $\mathcal{F}_{\mathsf{ZK}}$**

Upon receiving $(\mathsf{prove}, \mathcal{C}, w)$ from a prover $\mathcal{P}$ and $(\mathsf{verify}, \mathcal{C})$ from a verifier $\mathcal{V}$ where the same (boolean or arithmetic) circuit $\mathcal{C}$ is input by both parties, send $\mathsf{true}$ to $\mathcal{V}$ if $\mathcal{C}(w) = 1$; otherwise, send $\mathsf{false}$ to $\mathcal{V}$.

Figure 1: The zero-knowledge functionality.

**Functionality $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$**

**Initialize:** Upon receiving $\mathsf{init}$ from $\mathsf{P_A}$ and $\mathsf{P_B}$, sample $\Delta \leftarrow \mathbb{F}_{p^r}$ if $\mathsf{P_B}$ is honest or receive $\Delta \in \mathbb{F}_{p^r}$ from the adversary otherwise. Store global key $\Delta$, send $\Delta$ to $\mathsf{P_B}$, and ignore all subsequent $\mathsf{init}$ commands.

**Extend:** Upon receiving $(\mathsf{extend}, \ell)$ from $\mathsf{P_A}$ and $\mathsf{P_B}$, do:

1. If $\mathsf{P_B}$ is honest, sample $\mathsf{K}[\boldsymbol{x}] \leftarrow \mathbb{F}^{\ell}_{p^r}$. Otherwise, receive $\mathsf{K}[\boldsymbol{x}] \in \mathbb{F}^{\ell}_{p^r}$ from the adversary.
2. If $\mathsf{P_A}$ is honest, sample $\boldsymbol{x} \leftarrow \mathbb{F}^{\ell}_{p}$ and compute $\mathsf{M}[\boldsymbol{x}] := \mathsf{K}[\boldsymbol{x}] + \Delta \cdot \boldsymbol{x} \in \mathbb{F}^{\ell}_{p^r}$. Otherwise, receive $\boldsymbol{x} \in \mathbb{F}^{\ell}_{p}$ and $\mathsf{M}[\boldsymbol{x}] \in \mathbb{F}^{\ell}_{p^r}$ from the adversary, and then recompute $\mathsf{K}[\boldsymbol{x}] := \mathsf{M}[\boldsymbol{x}] - \Delta \cdot \boldsymbol{x}$.
3. Send $(\boldsymbol{x}, \mathsf{M}[\boldsymbol{x}])$ to $\mathsf{P_A}$ and $\mathsf{K}[\boldsymbol{x}]$ to $\mathsf{P_B}$.

**Global key query:** If $\mathsf{P_A}$ is corrupted, the adversary is allowed to make the following query *only once*:

1. Receive $(\mathsf{guess}, \Delta')$ from the adversary where $\Delta' \in \mathbb{F}_{p^r}$.
2. If $\Delta' = \Delta$, then send $\mathsf{success}$ to the adversary and continue. Otherwise, send $\mathsf{abort}$ to both parties and abort.

Figure 2: Functionality for subfield vector OLE.

for the sake of readability.) The online phase of our protocol relies on a functionality for *subfield vector oblivious linear evaluation* (sVOLE) [BCG+19a] that can be run by the prover and verifier in an offline phase; the corresponding functionality $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$ is given in Figure 2. This functionality allows two parties to generate a vector of authenticated values known to $\mathsf{P_A}$, as described in the previous section. Our sVOLE functionality follows the definition of Boyle et al. [BCG+19a], except that we allow the adversary to make a single global key query.

Other functionalities we rely on are given for reference in the appendix.

## 3 Online Phase of Our Zero-Knowledge Protocol

In Figure 3, we describe our zero-knowledge protocol $\Pi_{\mathsf{ZK}}$, which operates in the $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$-hybrid model. As noted in the Introduction, our protocol can be viewed as following a "GMW-style" approach to secure two-party computation using authenticated triples [NNOB12, DPSZ12]. In the secure-computation setting, evaluation of a multiplication gate requires two rounds of interaction, since the parties hold shares of the values on the input wires, but neither party knows those values. In the ZK setting, however, the prover $\mathcal{P}$ knows the values on all wires; thus, evaluation of a multiplication gate can be done without any interaction.

At a high level, our protocol consists of the following steps:

1. **Initialization.** The parties prepare authenticated values $\{[w_i]\}$ for the witness, and $\{[s_i]\}$ for each multiplication gate in the circuit. The parties also generate some number of authenticated

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

**Parameters and inputs:** The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold a circuit $\mathcal{C}$ over finite field $\mathbb{F}_p$ with $C$ multiplication gates; $\mathcal{P}$ holds a witness $w$ such that $\mathcal{C}(w) = 1$. Fix values $B, c$, and $r$, and let $\ell = C \cdot B + c$.

**Offline phase:**

1. $\mathcal{P}$ (act as $\mathsf{P_A}$) and $\mathcal{V}$ (act as $\mathsf{P_B}$) send $\mathsf{init}$ to $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, which returns a uniform $\Delta \in \mathbb{F}_{p^r}$ to $\mathcal{V}$.
2. $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{extend}, |\mathcal{I}_{\mathsf{in}}| + 3\ell + C)$ to $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, which returns authenticated values $\{[\lambda_i]\}_{i \in \mathcal{I}_{\mathsf{in}}}$, $\{([x_i], [y_i], [r_i])\}_{i \in [\ell]}$, and $\{[s_i]\}_{i \in [C]}$ to the parties. (If $\mathcal{V}$ receives $\mathsf{abort}$ from $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, then it aborts.)
3. For $i \in [\ell]$, $\mathcal{P}$ sends $d_i := x_i \cdot y_i - r_i \in \mathbb{F}_p$ to $\mathcal{V}$, and then both parties compute $[z_i] := [r_i] + d_i$.

**Online phase:**

4. For $i \in \mathcal{I}_{\mathsf{in}}$, $\mathcal{P}$ sends $\Lambda_i := w_i - \lambda_i \in \mathbb{F}_p$ to $\mathcal{V}$, and then both parties compute $[w_i] := [\lambda_i] + \Lambda_i$.
5. For each gate $(\alpha, \beta, \gamma, T) \in \mathcal{C}$, in topological order:
   (a) If $T = \mathsf{Add}$, the two parties locally compute $[w_\gamma] := [w_\alpha] + [w_\beta]$.
   (b) If $T = \mathsf{Mult}$ and this is the $i$th multiplication gate, $\mathcal{P}$ sends $d := w_\alpha \cdot w_\beta - s_i \in \mathbb{F}_p$ to $\mathcal{V}$, and then both parties compute $[w_\gamma] := [s_i] + d$.
6. $\mathcal{V}$ samples a random permutation $\pi$ on $\{1, \ldots, \ell\}$ and sends it to $\mathcal{P}$. The two parties use $\pi$ to permute the $\{([x_i], [y_i], [z_i])\}_{i \in [\ell]}$ obtained in step 3.
7. For the $i$th multiplication gate $(\alpha, \beta, \gamma, \mathsf{Mult})$, where the parties obtained $([w_\alpha], [w_\beta], [w_\gamma])$ in step 5, do the following for $j = 1, \ldots, B$:
   (a) Let $([x], [y], [z])$ be the $\big((i-1)B + j\big)$th authenticated triple (after applying $\pi$ in step 6).
   (b) The parties run $\delta_\alpha := \mathsf{Open}([w_\alpha] - [x])$ and $\delta_\beta := \mathsf{Open}([w_\beta] - [y])$. The parties then compute $[\mu] := [z] - [w_\gamma] + \delta_\beta \cdot [x] + \delta_\alpha \cdot [y] + \delta_\alpha \cdot \delta_\beta$, and finally run $\mathsf{CheckZero}([\mu])$.
8. For each of the remaining $c$ authenticated triples, say $([x], [y], [z])$, the parties run $x := \mathsf{Open}([x])$ and $y := \mathsf{Open}([y])$. They also compute $[\nu] := [z] - x \cdot y$ and then run $\mathsf{CheckZero}([\nu])$.
9. For the single output wire $o$ with authenticated value $[w_o]$, the parties run $\mathsf{CheckZero}([w_o] - 1)$.

Figure 3: Zero-knowledge proof in the $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$-hybrid model.

multiplication triples $\{([x_i], [y_i], [z_i])\}$; a malicious prover may cause some or all of these triples to be incorrect (i.e., $z_i \neq x_i \cdot y_i$).

2. **Circuit evaluation.** Starting with the authenticated values $\{[w_i]\}$ at the input wires, the parties inductively compute authenticated values for all the wires in the circuit. For addition gates, this is easy. For the $i$th multiplication gate, the prover uses $[s_i]$ to enable the verifier to compute its component of the authenticated value for the output wire without revealing information about the values on the input wires. Specifically, given authenticated values $[w_\alpha], [w_\beta]$ on the input wires to the $i$th multiplication gate, the prover sends $w_\alpha \cdot w_\beta - s_i$ to the verifier; the prover and verifier then compute

$$[w_\gamma] := [s_i] + (w_\alpha \cdot w_\beta - s_i)$$

as the authenticated value of the output wire. All communication here is from the prover to the verifier, so the entire circuit can be evaluated using only one round of communication.

Once the parties have an authenticated value $[w_o]$ for the output wire, the prover simply opens that value and the verifier checks that it is equal to 1.

3. **Verifying correct behavior.** So far, nothing has prevented a malicious prover from cheating. To prevent this, the verifier needs to check the behavior of the prover at each multiplication gate using the initial set of authenticated multiplication triples the parties generated. This can be done in various ways. In the protocol as described in Figure 3, which works for circuits over an arbitrary field $\mathbb{F}_p$, the verifier checks the behavior of the prover as follows (adapting [FLNW17, ABF+17]):

   - The verifier checks a random subset of the authenticated multiplication triples to make sure they are correctly formed. For an authenticated multiplication triple $[x], [y], [z]$, this can be done by having the prover run $\mathsf{Open}([x])$ and $\mathsf{Open}([y])$ followed by $\mathsf{CheckZero}([z] - x \cdot y)$.
   - The verifier then uses the remaining authenticated multiplication triples to check that each multiplication gate was computed correctly. For a multiplication gate with authenticated value $[w_\alpha], [w_\beta]$ on the input wires and $[w_\gamma]$ on the output wire, the relation $w_\gamma = w_\alpha w_\beta$ can be checked using an authenticated multiplication triple $[x], [y], [z]$ by having the prover run $\delta_\alpha := \mathsf{Open}([w_\alpha] - [x])$ and $\delta_\beta := \mathsf{Open}([w_\beta] - [y])$, followed by

     $$\mathsf{CheckZero}\left([z] - [w_\gamma] + \delta_\beta \cdot [x] + \delta_\alpha \cdot [y] + \delta_\alpha \cdot \delta_\beta\right).$$

     Each multiplication gate is checked in this way using $B$ authenticated multiplication triples.

   Note that the checks for the openings of all the authenticated values (i.e., all the executions of $\mathsf{Open}$ and $\mathsf{CheckZero}$) can be batched together at the end of the protocol.

   At the end of the following section, we describe a more efficient approach for verifying correct behavior in the case of large $p$.

### 3.1 Proof of Security

Before giving the proof of security for protocol $\Pi_{\mathsf{ZK}}$, we analyze the procedure used to check correctness of the multiplication gates. Consider some multiplication gate with authenticated values $[w_\alpha]$, $[w_\beta]$ on the input wires and $[w_\gamma]$ on the output wire. If $\mathcal{P}$ cheated, so $w_\gamma \neq w_\alpha \cdot w_\beta$, then this cheating will be detected in step 7 unless all $B$ of the multiplication triples $([x], [y], [z])$ used to check that gate are incorrect. (We ignore for now the possibility that $\mathcal{P}$ is able to successfully cheat when running $\mathsf{Open}$.) But if too many of the initial multiplication triples are incorrect, then there is a high probability that $\mathcal{P}$ will be caught in step 8. We can analyze the overall probability with which a cheating $\mathcal{P}$ can successfully evade detection by considering an abstract "balls-and-bins" game $\mathsf{BBG}$ with an adversary $\mathcal{A}$, which is based on a similar game considered previously in the context of secure three-party computation [FLNW17]. Specifically, the game proceeds as follows:

1. $\mathcal{A}$ prepares $\ell = CB + c$ balls $\mathcal{B}_1, \ldots, \mathcal{B}_\ell$, each of which is either $\mathsf{good}$ or $\mathsf{bad}$. $\mathcal{A}$ also prepares $C$ bins, each of which is either $\mathsf{good}$ or $\mathsf{bad}$. The balls $\{\mathcal{B}_i\}_{i \in [\ell]}$ correspond to the triples $\{([x_i], [y_i], [z_i])\}_{i \in [\ell]}$ defined in step 3 of the protocol, and the bins correspond to the triples $\{([w_\alpha], [w_\beta], [w_\gamma])\}$ defined for the multiplication gates during the circuit evaluation. A $\mathsf{bad}$ ball/bin is an incorrect triple, while a $\mathsf{good}$ ball/bin is a correct triple.

2. Then, $c$ random balls are chosen. If any of the chosen balls is $\mathsf{bad}$, $\mathcal{A}$ loses. Otherwise, the game proceeds to the next step.

3. The remaining $CB$ balls are randomly partitioned into the $C$ bins, with each bin receiving exactly $B$ balls.

4. We say that a bin is fully good (resp., fully bad) if it is labeled good and all the balls inside it are good (resp., labeled bad and all the balls inside it are bad). $\mathcal{A}$ wins if and only if there exists at least one bin that is fully bad, and all other bins are either fully good or fully bad.

**Lemma 2.** *Assume $c \geq B$. Then $\mathcal{A}$ wins the above game with probability at most $\binom{CB+c}{B}^{-1}$.*

*Proof.* Assume that $\mathcal{A}$ makes $m$ bins bad for $1 \leq m \leq C$. It is easy to see that $\mathcal{A}$ can only possibly win if exactly $mB$ balls among $\mathcal{B}_1, \ldots, \mathcal{B}_\ell$ are bad, and they are exactly placed in the $m$ bins that are bad. We compute the probability that $\mathcal{A}$ wins for some fixed $m$.

Since exactly $mB$ balls of the $\ell = CB + c$ balls are bad, the probability that none of the bad balls is chosen in step 2 of the game is exactly

$$\frac{\binom{\ell - mB}{c}}{\binom{\ell}{c}} = \frac{(\ell - mB)! \cdot (\ell - c)!}{\ell! \cdot (\ell - mB - c)!} = \frac{(CB + c - mB)! \cdot (CB)!}{(CB + c)! \cdot (CB - mB)!}.$$

Assume that this occurs. We are left with $\ell - c = CB$ balls, of which $mB$ are bad. The probability that $B$ bad balls are placed in each bad bin is

$$\frac{(mB)! \cdot (CB - mB)!}{(CB)!}.$$

Thus, the probability that $\mathcal{A}$ wins is exactly

$$\frac{\binom{\ell - mB}{c}}{\binom{\ell}{c}} \cdot \frac{(mB)! \cdot (CB - mB)!}{(CB)!} = \frac{(CB + c - mB)! \cdot (mB)!}{(CB + c)!} = \binom{CB + c}{mB}^{-1}.$$

When $c \geq B$ (and $1 \leq m \leq C$), this is maximized when $m = 1$. □

We now prove security of $\Pi_{\mathsf{ZK}}$.

**Theorem 1.** *Let $c \geq B$. Protocol $\Pi_{\mathsf{ZK}}$ UC-realizes $\mathcal{F}_{\mathsf{ZK}}$ in the $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$-hybrid model. In particular, no environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution, except with probability at most $\binom{CB+c}{B}^{-1} + p^{-r} + \varepsilon_{\mathsf{open}}$.*

*Proof.* We first consider the case of a malicious prover (i.e., soundness) and then consider the case of a malicious verifier (i.e., zero knowledge). In each case, we construct a PPT simulator $\mathcal{S}$ given access to $\mathcal{F}_{\mathsf{ZK}}$, and running the PPT adversary $\mathcal{A}$ as a subroutine while emulating functionality $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ for $\mathcal{A}$. We always implicitly assume that $\mathcal{S}$ passes all communication between $\mathcal{A}$ and $\mathcal{Z}$.

**Malicious prover.** $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ for $\mathcal{A}$ by choosing uniform $\Delta \in \mathbb{F}_{p^r}$ and recording all the values $\{\lambda_i\}_{i \in \mathcal{I}_{\mathsf{in}}}$, $\{(x_i, y_i, r_i)\}_{i \in [\ell]}$, and $\{s_i\}_{i \in [C]}$, and their corresponding MAC tags, sent to $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ by $\mathcal{A}$. These values define corresponding keys in the natural way.

2. If $\mathcal{A}$ makes a global key query $(\mathsf{guess}, \Delta')$, then $\mathcal{S}$ checks if $\Delta' = \Delta$. If not, $\mathcal{S}$ sends abort to $\mathcal{A}$, sends $(\mathsf{prove}, \mathcal{C}, \bot)$ to $\mathcal{F}_{\mathsf{ZK}}$, and stops. Otherwise, $\mathcal{S}$ aborts.

3. When $\mathcal{A}$ sends $\{\Lambda_i\}_{i \in \mathcal{I}_{\mathsf{in}}}$ in step 4, $\mathcal{S}$ sets $w_i := \lambda_i + \Lambda_i$ for $i \in \mathcal{I}_{\mathsf{in}}$.

4. $\mathcal{S}$ runs the rest of the protocol as an honest verifier, using $\Delta$ and the keys defined in the first step. If the honest verifier outputs false, then $\mathcal{S}$ sends $(\mathsf{prove}, \mathcal{C}, \bot)$ to $\mathcal{F}_{\mathsf{ZK}}$ and aborts. If the honest verifier outputs true, $\mathcal{S}$ sends $(\mathsf{prove}, \mathcal{C}, w)$ to $\mathcal{F}_{\mathsf{ZK}}$ where $w$ is as defined above.

We assume $\mathcal{A}$ does not correctly guess $\Delta$; this is true except with probability at most $p^{-r}$. It is then clear that the view of $\mathcal{A}$ is perfectly simulated by $\mathcal{S}$, and the simulated verifier run by $\mathcal{S}$ has the same output distribution as the verifier in the real protocol execution. Whenever the verifier simulated by $\mathcal{S}$ outputs $\mathsf{false}$, the real verifier outputs $\mathsf{false}$ as well (since $\mathcal{S}$ sends $\perp$ to $\mathcal{F}_{\mathsf{ZK}}$). It thus only remains to show that, except with negligible probability, if the simulated verifier run by $\mathcal{S}$ outputs $\mathsf{true}$, then the witness $w$ sent by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{ZK}}$ satisfies $\mathcal{C}(w) = 1$. Below, we show that if $\mathcal{C}(w) = 0$ then the probability that the simulated verifier outputs $\mathsf{true}$ is at most $\binom{CB+c}{B}^{-1} + \varepsilon_{\mathsf{open}}$.

If $\mathcal{C}(w) = 0$ then either $w_o = 0$ or else at least one of the triples $\{([w_\alpha], [w_\beta], [w_\gamma])\}$ defined at the multiplication gates during the circuit evaluation must be incorrect. In the former case, the probability that $\mathcal{P}$ can succeed when running $\mathsf{CheckZero}(w_o)$ is at most $\varepsilon_{\mathsf{open}}$. In the latter case, Lemma 2 shows that the probability that $\mathcal{A}$ avoids being "caught" in steps 6–8 is at most $\binom{CB+c}{B}^{-1}$; if $\mathcal{A}$ is caught, then it succeeds in opening some incorrect value with probability at most $\varepsilon_{\mathsf{open}}$. This completes the proof for the case of a malicious prover.

**Malicious verifier.** If $\mathcal{S}$ receives $\mathsf{false}$ from $\mathcal{F}_{\mathsf{ZK}}$, then it simply aborts. Otherwise, $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ for $\mathcal{A}$ by recording the global key $\Delta$, and the keys for all the authenticated values, sent to the functionality by $\mathcal{A}$. Then, $\mathcal{S}$ samples uniform values for $\{\lambda_i\}_{i\in\mathcal{I}_{\mathsf{in}}}$, $\{(x_i, y_i, r_i)\}_{i\in[\ell]}$, and $\{s_i\}_{i\in[C]}$, and computes their corresponding MAC tags in the natural way.

2. $\mathcal{S}$ executes steps 3–8 of protocol $\Pi_{\mathsf{ZK}}$ by simulating the honest prover with input $w = 0^{|\mathcal{I}_{\mathsf{in}}|}$.

3. In step 9, $\mathcal{S}$ computes $\mathsf{K}[w_o]$ (based on the keys sent to $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ by $\mathcal{A}$) and then sets $\mathsf{M}[w_o] := \mathsf{K}[w_o] + \Delta$. Finally, it uses $\mathsf{M}[w_o]$ to run $\mathsf{CheckZero}([w_o] - 1)$ with $\mathcal{A}$.

The view of adversary $\mathcal{A}$ simulated by $\mathcal{S}$ is distributed identically to its view in the real protocol execution. This completes the proof. □

**Verifying correct behavior for large $p$.** For large $p$, we can use a different procedure for checking correctness of the multiplication gates that can be viewed as a simplified version of the check used by SPDZ [DPSZ12]. Specifically, the parties now prepare a *single* multiplication triple $[x], [y], [z]$ per multiplication gate. To check correctness of a multiplication gate with authenticated values $[w_\alpha]$, $[w_\beta]$ on the input wires, and $[w_\gamma]$ on the output wire, the verifier sends a uniform $\eta \leftarrow \mathbb{F}_p$ to the prover, who responds by running $\delta_\alpha := \mathsf{Open}(\eta \cdot [w_\alpha] - [x])$ and $\delta_\beta := \mathsf{Open}([w_\beta] - [y])$, followed by

$$\mathsf{CheckZero}([z] - \eta \cdot [w_\gamma] + \delta_\beta \cdot [x] + \delta_\alpha \cdot [y] + \delta_\alpha \cdot \delta_\beta).$$

This has soundness error $1/p + \varepsilon_{\mathsf{open}}$. To see this, say $w_\gamma = w_\alpha w_\beta + \Delta_w$ with $\Delta_w \neq 0$, and let $z = xy + \Delta_z$. Then $z - \eta \cdot w_\gamma + \delta_\beta \cdot x + \delta_\alpha \cdot y + \delta_\alpha \cdot \delta_\beta = 0$ iff $\eta = \Delta_z/\Delta_w$. Note that this checking procedure can be done for all multiplication gates in parallel using a single value $\eta$, and the overall soundness error remains unchanged.

# 4 Subfield VOLE

The online phase of out zero-knowledge protocol relies on sVOLE [BCGI18, BCG+19b] to generate authenticated values. Boyle et al. [BCGI18] proposed an sVOLE protocol with sublinear communication complexity, but their protocol relies on generic secure two-party computation (2PC) and is thus not very efficient. Subsequent work either considered sVOLE for general fields but with

semi-honest security [BCG+19b, SGRR19], or focused on concrete efficiency but only for the case of binary fields [BCG+19a, YWL+20].

Here, we extend the work of Kang et al. [YWL+20] to arbitrary fields. Moreover, although Kang et al. only consider security in the stand-alone model, we prove our protocol secure in the UC framework. To do so, we consider a slightly relaxed sVOLE functionality that allows for a single global key query; this has negligible impact when using sVOLE in our ZK proof.

**Overview.** In Appendix B, we present an sVOLE protocol inspired by the work of Keller et al. [KOS15, KOS16]. Although that protocol technically suffices for our ZK protocol, we can obtain better efficiency using "sVOLE extension" (by analogy with OT extension), by which we extend a small number of "base" sVOLE correlations into a larger number of sVOLE correlations. Toward this end, in Section 4.1, we show how to construct a protocol for *single-point* sVOLE (spsVOLE) from sVOLE, where spsVOLE is like sVOLE except that the vector of authenticated values has only a *single* non-zero entry. Finally, in Section 4.2, we present an efficient protocol for "sVOLE extension" using spsVOLE as a subroutine and relying on a variant of the *learning parity with noise* (LPN) assumption. We provide intuition for each of these protocols in the relevant section.

## 4.1 Single-Point Subfield VOLE

**Functionality $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$**

**Initialize:** Upon receiving init from $\mathsf{P_A}$ and $\mathsf{P_B}$, sample $\Delta \leftarrow \mathbb{F}_{p^r}$ if $\mathsf{P_B}$ is honest or receive $\Delta \in \mathbb{F}_{p^r}$ from the adversary otherwise. Store global key $\Delta$, send $\Delta$ to $\mathsf{P_B}$, and ignore all subsequent init commands.

**Extend:** Upon receiving $(\mathsf{sp\text{-}extend}, n)$, where $n = 2^h$ for some $h \in \mathbb{N}$, from $\mathsf{P_A}$ and $\mathsf{P_B}$, do:

1. If $\mathsf{P_B}$ is honest, sample $\boldsymbol{v} \leftarrow \mathbb{F}^n_{p^r}$. Otherwise, receive $\boldsymbol{v} \in \mathbb{F}^n_{p^r}$ from the adversary.
2. If $\mathsf{P_A}$ is honest, then sample a uniform $\boldsymbol{u} \in \mathbb{F}^n_p$ with exactly one non-zero entry, and compute $\boldsymbol{w} := \boldsymbol{v} + \Delta \cdot \boldsymbol{u} \in \mathbb{F}^n_{p^r}$. Let $\alpha \in [0, n)$ be the index of the non-zero entry of $\boldsymbol{u}$. Otherwise, receive $\boldsymbol{u} \in \mathbb{F}^n_p$ (with at most one non-zero entry) and $\boldsymbol{w} \in \mathbb{F}^n_{p^r}$ from the adversary, and set $\boldsymbol{v} := \boldsymbol{w} - \Delta \cdot \boldsymbol{u}$.
3. If $\mathsf{P_B}$ is corrupted, receive $I \subseteq [0, n)$ from the adversary. If $\alpha \in I$, send success to $\mathsf{P_B}$ and continue. Otherwise, send abort to $\mathsf{P_A}$; when $\mathsf{P_A}$ responds with abort, send abort to $\mathsf{P_B}$ and stop.
4. Send $(\boldsymbol{u}, \boldsymbol{w})$ to $\mathsf{P_A}$ and $\boldsymbol{v}$ to $\mathsf{P_B}$.

**Global key query:** If $\mathsf{P_A}$ is corrupted, the adversary is allowed to make the following query only once:

1. Receive $(\mathsf{guess}, \Delta')$ from the adversary where $\Delta' \in \mathbb{F}_{p^r}$.
2. If $\Delta' = \Delta$, then send success to the adversary and continue. Otherwise, send abort to both parties and abort.

Figure 4: Functionality for single-point sVOLE.

We present the single-point sVOLE (spsVOLE) functionality $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$ in Figure 4, where the length $n = 2^h$ of the vectors is assumed to be a power of two for simplicity. $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$ is a version of sVOLE where the vector of authenticated values contains exactly one nonzero entry.

**Overview of our protocol.** In Figure 5, we present protocol $\Pi^{p,r}_{\mathsf{spsVOLE}}$ that UC-realizes $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$ in the $(\mathcal{F}_{\mathsf{OT}}, \mathcal{F}^{p,r}_{\mathsf{sVOLE}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model, where $\mathcal{F}_{\mathsf{EQ}}$ corresponds to a weak equality test that reveals $\mathsf{P_A}$'s input to $\mathsf{P_B}$. (See Appendix A.) The protocol can be conceptually divided into two steps: first, the parties run a semi-honest protocol for generating a vector of authenticated values $[\boldsymbol{u}]$ having

a single nonzero entry; then, a consistency check is performed to detect malicious behavior. We explain both steps in what follows.

$\mathsf{P_A}$ begins by choosing a uniform $\beta \in \mathbb{F}_p^*$ and a uniform index $\alpha$. Letting $\boldsymbol{u} \in \mathbb{F}_p^n$ be the vector that is 0 everywhere except that $\boldsymbol{u}[\alpha] = \beta$, the goal is for the parties to generate $[\boldsymbol{u}]$. That is, they want $\mathsf{P_A}$ to hold $\boldsymbol{w} \in \mathbb{F}_{p^r}^n$ and $\mathsf{P_B}$ to hold $\boldsymbol{v} \in \mathbb{F}_{p^r}^n$ such that $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$. To do so, the parties begin by generating the authenticated value $[\beta]$; this is easy to do using a call to $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$. Next, they use a subroutine [BGI15, BGI16, BCG+17] based on the GGM construction [GGM86] to enable $\mathsf{P_B}$ to generate $\boldsymbol{v} \in \mathbb{F}_{p^r}^n$ while allowing $\mathsf{P_A}$ to learn all the components of that vector except for $\boldsymbol{v}[\alpha]$. This is done in the following way. Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ and $G' : \{0,1\}^\kappa \to \mathbb{F}_{p^r}^2$ be pseudorandom generators. $\mathsf{P_B}$ chooses uniform $s \in \{0,1\}^\kappa$ and computes all values in a GGM tree of depth $h$ with $s$ at the root: That is, letting $s_j^i$ denote the value at the $j$th node on the $i$th level of the tree, $\mathsf{P_B}$ defines $s_0^0 := s$ and then for $i = 1, \ldots, h-1$ and $j \in [0, 2^{i-1})$ computes $\left(s_{2j}^i, s_{2j+1}^i\right) := G(s_j^{i-1})$; finally, $\mathsf{P_B}$ computes a vector $\boldsymbol{v}$ at the leaves as $(\boldsymbol{v}[2j], \boldsymbol{v}[2j+1]) := G'(s_j^{h-1})$ for $j \in [0, 2^{h-1})$. Next, $\mathsf{P_B}$ lets $K_0^i$ (resp., $K_1^i$) be the XOR of the values at the even (resp., odd) nodes on level $i$. (When $i = h$ we replace XOR with addition in $\mathbb{F}_{p^r}$.) We write $\left(\{\boldsymbol{v}[j]\}_{j=0}^{n-1}, \{(K_0^i, K_1^i)\}_{i=1}^h\right) \leftarrow \mathsf{GGM}(1^n, s)$ to denote this computation done by $\mathsf{P_B}$. It is easily verified that if $\mathsf{P_A}$ is given $\{K_{\bar{\alpha}_i}^i\}_{i=1}^h$ (where $\bar{\alpha}_i$ is the complement of the $i$th bit of $\alpha$), then $\mathsf{P_A}$ can compute $\{\boldsymbol{v}[j]\}_{j \neq \alpha}$, while $\boldsymbol{v}[\alpha]$ remains computationally indistinguishable from uniform given $\mathsf{P_A}$'s view. ($\mathsf{P_A}$ can obtain $\{K_{\bar{\alpha}_i}^i\}_{i=1}^h$ using $h$ OT invocations.) We denote the resulting computation of $\mathsf{P_A}$ by $\{\boldsymbol{v}[j]\}_{j \neq \alpha} \leftarrow \mathsf{GGM}'(\alpha, \{K_{\bar{\alpha}_i}^i\}_{i=1}^h)$.

Following the above, $\mathsf{P_A}$ sets $\boldsymbol{w}[i] := \boldsymbol{v}[i]$ for $i \neq \alpha$. Note that $\boldsymbol{w}[i] = \boldsymbol{v}[i] + \Delta \cdot \boldsymbol{u}[i]$ for $i \neq \alpha$ (since $\boldsymbol{u}[i] = 0$ for $i \neq \alpha$), so all that remains is for $\mathsf{P_A}$ to obtain the missing value $\boldsymbol{w}[\alpha] = \boldsymbol{v}[\alpha] + \Delta \cdot \beta$ (without revealing $\alpha, \beta$ to $\mathsf{P_B}$). Recall the parties already hold $[\beta]$, meaning that $\mathsf{P_A}$ holds $\mathsf{M}[\beta]$ and $\mathsf{P_B}$ holds $\mathsf{K}[\beta]$ with $\mathsf{M}[\beta] = \mathsf{K}[\beta] + \Delta \cdot \beta$. So if $\mathsf{P_B}$ sends $\mathsf{K}[\beta] - \sum_i \boldsymbol{v}[i]$, then $\mathsf{P_A}$ can compute the missing value as

$$\begin{aligned} \boldsymbol{w}[\alpha] &:= \mathsf{M}[\beta] - (\mathsf{K}[\beta] - \textstyle\sum_i \boldsymbol{v}[i]) - \sum_{i \neq \alpha} \boldsymbol{v}[i] \\ &= \mathsf{M}[\beta] - \mathsf{K}[\beta] + \boldsymbol{v}[\alpha] = \boldsymbol{v}[\alpha] + \Delta \cdot \beta. \end{aligned}$$

This completes the "semi-honest" portion of the protocol.

To verify correct behavior, we generalize the approach of Kang et al. [YWL+20] that applies only to the case $p = 2$. We want to verify that $\boldsymbol{w}[i] = \boldsymbol{v}[i]$ for $i \neq \alpha$, and $\boldsymbol{w}[\alpha] = \boldsymbol{v}[\alpha] + \Delta \cdot \beta$. Intuitively, the parties do this by having $\mathsf{P_A}$ choose uniform $\chi_0, \ldots, \chi_{n-1} \in \mathbb{F}_{p^r}$ and then checking that

$$\sum_i \chi_i \cdot \boldsymbol{w}[i] = \sum_i \chi_i \cdot \boldsymbol{v}[i] + \Delta \cdot \beta \cdot \chi_\alpha.$$

Of course, this must be done without revealing $\alpha, \beta$ to $\mathsf{P_B}$. To do so, $\mathsf{P_A}$ and $\mathsf{P_B}$ use $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ to compute $Z, Y \in \mathbb{F}_{p^r}$, respectively, such that $Z = Y + \Delta \cdot \beta \cdot \chi_\alpha$. (We discuss below how this is done.) They then use $\mathcal{F}_{\mathsf{EQ}}$ to check if $V_{\mathsf{A}} = \sum_{i=0}^{n-1} \chi_i \cdot \boldsymbol{w}[i] - Z$ is equal to $V_{\mathsf{B}} = \sum_{i=0}^{n-1} \chi_i \cdot \boldsymbol{v}[i] - Y$.

To complete the description, we show how $\mathsf{P_A}$, $\mathsf{P_B}$ can generate $Z, Y$ (held by $\mathsf{P_A}$, $\mathsf{P_B}$, respectively) such that $Z = Y + \Delta \cdot \beta \cdot \chi_\alpha$. Although this looks like an authenticated value $[\beta \cdot \chi_\alpha]$, note that $\beta \cdot \chi_\alpha$ lies in $\mathbb{F}_{p^r}$ rather than $\mathbb{F}_p$. $\mathsf{P_A}$ views $\chi_\alpha \in \mathbb{F}_{p^r}$ as a vector $\boldsymbol{\chi}_\alpha = (\chi_{\alpha,0}, \ldots, \chi_{\alpha,r-1}) \in \mathbb{F}_p^r$ (namely, $\chi_\alpha = \sum_i \chi_{\alpha,i} \cdot g^i$), and then the two parties use $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ to generate the vector of authenticated values $[\beta \cdot \boldsymbol{\chi}_\alpha]$. This means $\mathsf{P_A}$ holds $\boldsymbol{z}$ and $\mathsf{P_B}$ holds $\boldsymbol{y}$ such that $\boldsymbol{z} = \boldsymbol{y} + \Delta \cdot \beta \cdot \boldsymbol{\chi}_\alpha$. Letting

**Protocol $\Pi^{p,r}_{\mathsf{spsVOLE}}$**

**Initialize:** This procedure is executed only once.

1. $\mathsf{P_A}$ and $\mathsf{P_B}$ send $\mathsf{init}$ to $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$, which returns $\Delta \in \mathbb{F}_{p^r}$ to $\mathsf{P_B}$.

**Extend:** This procedure can be run multiple times. On input $n = 2^h$, the parties do:

2. $\mathsf{P_A}$ and $\mathsf{P_B}$ send $(\mathsf{extend}, 1)$ to $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$, which returns $(a, c) \in \mathbb{F}_p \times \mathbb{F}_{p^r}$ to $\mathsf{P_A}$ and $b \in \mathbb{F}_{p^r}$ to $\mathsf{P_B}$ such that $c = b + \Delta \cdot a$. Then, $\mathsf{P_A}$ samples $\beta \leftarrow \mathbb{F}_p^*$, sets $\delta := c$, and sends $a' := \beta - a \in \mathbb{F}_p$ to $\mathsf{P_B}$, who computes $\gamma := b - \Delta \cdot a'$. Note that $\delta = \gamma + \Delta \cdot \beta \in \mathbb{F}_{p^r}$.

   $\mathsf{P_A}$ samples $\alpha \leftarrow [0, n)$ and defines $\boldsymbol{u}$ to be a vector of length $n$ that is 0 everywhere except $\boldsymbol{u}[\alpha] := \beta$.

3. $\mathsf{P_B}$ samples $s \leftarrow \{0,1\}^\kappa$, runs $\left(\{v_j\}_{j=0}^{n-1}, \{(K_0^i, K_1^i)\}_{i=1}^h\right) \leftarrow \mathsf{GGM}(1^n, s)$, and sets $\{\boldsymbol{v}[j] := v_j\}_{j=0}^{n-1}$.

   $\mathsf{P_A}$ sets $\alpha_i$ as the $i$th bit of the binary representation of $\alpha$. For $i \in [h]$, $\mathsf{P_A}$ sends $\bar{\alpha}_i \in \{0,1\}$ to $\mathcal{F}_{\mathsf{OT}}$ and $\mathsf{P_B}$ sends $(K_0^i, K_1^i)$ to $\mathcal{F}_{\mathsf{OT}}$, which returns $K^i_{\bar{\alpha}_i}$ to $\mathsf{P_A}$. Then $\mathsf{P_A}$ runs $\{v_j\}_{j \neq \alpha} \leftarrow \mathsf{GGM}'(\alpha, \{K^i_{\bar{\alpha}_i}\}_{i=1}^h)$.

4. $\mathsf{P_B}$ sends $d := \gamma - \sum_{i \in [0,n)} \boldsymbol{v}[i] \in \mathbb{F}_{p^r}$ to $\mathsf{P_A}$. Then, $\mathsf{P_A}$ defines $\boldsymbol{w}$ to be a vector of length $n$ with $\boldsymbol{w}[i] := v_i$ for $i \neq \alpha$ and $\boldsymbol{w}[\alpha] := \delta - \left(d + \sum_{i \neq \alpha} \boldsymbol{w}[i]\right)$. Note that $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$.

**Consistency check:**

5. Both parties send $(\mathsf{extend}, r)$ to $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$, which returns $(\boldsymbol{x}, \boldsymbol{z}) \in \mathbb{F}_p^r \times \mathbb{F}_{p^r}^r$ to $\mathsf{P_A}$ and $\boldsymbol{y}^* \in \mathbb{F}_{p^r}^r$ to $\mathsf{P_B}$ such that $\boldsymbol{z} = \boldsymbol{y}^* + \Delta \cdot \boldsymbol{x}$.

6. $\mathsf{P_A}$ samples $\chi_i \leftarrow \mathbb{F}_{p^r}$ for $i \in [0, n)$, and writes $\chi_\alpha = \sum_{i=0}^{r-1} \chi_{\alpha,i} \cdot g^i$. Let $\boldsymbol{\chi}_\alpha = (\chi_{\alpha,0}, \ldots, \chi_{\alpha,r-1}) \in \mathbb{F}_p^r$. $\mathsf{P_A}$ then computes $\boldsymbol{x}^* := \beta \cdot \boldsymbol{\chi}_\alpha - \boldsymbol{x} \in \mathbb{F}_p^r$ and sends $\{\chi_i\}_{i \in [0,n)}$ and $\boldsymbol{x}^*$ to $\mathsf{P_B}$, who computes $\boldsymbol{y} := \boldsymbol{y}^* - \Delta \cdot \boldsymbol{x}^* \in \mathbb{F}_{p^r}^r$.

7. $\mathsf{P_A}$ sets $Z := \sum_{i=0}^{r-1} \boldsymbol{z}[i] \cdot g^i$ and $V_{\mathsf{A}} := \sum_{i \in [0,n)} \chi_i \cdot \boldsymbol{w}[i] - Z$, while $\mathsf{P_B}$ sets $Y := \sum_{i=0}^{r-1} \boldsymbol{y}[i] \cdot g^i$ and $V_{\mathsf{B}} := \sum_{i \in [0,n)} \chi_i \cdot \boldsymbol{v}[i] - Y$. Then $\mathsf{P_A}$ sends $V_{\mathsf{A}}$ to $\mathcal{F}_{\mathsf{EQ}}$, and $\mathsf{P_B}$ sends $V_{\mathsf{B}}$ to $\mathcal{F}_{\mathsf{EQ}}$. If either party receives $\mathsf{false}$ or $\mathsf{abort}$ from $\mathcal{F}_{\mathsf{EQ}}$, it aborts.

8. $\mathsf{P_A}$ outputs $(\boldsymbol{u}, \boldsymbol{w})$ and $\mathsf{P_B}$ outputs $\boldsymbol{v}$.

Figure 5: Single-point sVOLE protocol in the $(\mathcal{F}_{\mathsf{OT}}, \mathcal{F}^{p,r}_{\mathsf{sVOLE}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model.

$Z = \sum_i \boldsymbol{z}[i] \cdot g^i$ and $Y = \sum_i \boldsymbol{y}[i] \cdot g^i$, we then have

$$\begin{aligned} Z = \sum_i \boldsymbol{z}[i] \cdot g^i &= \sum_i (\boldsymbol{y}[i] + \Delta \cdot \beta \cdot \boldsymbol{\chi}_\alpha[i]) \cdot g^i \\ &= \sum_i \boldsymbol{y}[i] \cdot g^i + \Delta \cdot \beta \cdot \sum_i \boldsymbol{\chi}_\alpha[i] \cdot g^i \\ &= Y + \Delta \cdot \beta \cdot \chi_\alpha, \end{aligned}$$

as desired.

We remark that the consistency check allows a malicious $\mathsf{P_A}$ to attempt to guess $\Delta$, and allows a malicious $\mathsf{P_B}$ to attempt to guess a subset in which the index $\alpha$ lies. (This will become evident in the proof of security below.) The possibility of such guesses is incorporated into the ideal functionality $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$, and we show later that (as in prior work [BCG+19a, YWL+20]) this leakage is harmless when $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$ is used as a subroutine in our final sVOLE protocol.

**Theorem 2.** *If $G$ and $G'$ are pseudorandom generators, then $\Pi^{p,r}_{\mathsf{spsVOLE}}$ UC-realizes $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$ in the $(\mathcal{F}_{\mathsf{OT}}, \mathcal{F}^{p,r}_{\mathsf{sVOLE}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model. In particular, no PPT environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution, except with probability at most $1/p^r + \mathsf{negl}(\kappa)$.*

*Proof.* We first consider the case of a malicious $\mathsf{P_A}$ and then consider the case of a malicious $\mathsf{P_B}$. In each case, we construct a PPT simulator $\mathcal{S}$ given access to $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$ that runs the PPT adversary $\mathcal{A}$ as a subroutine, and emulates $\mathcal{F}_{\mathsf{OT}}$,$\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, and $\mathcal{F}_{\mathsf{EQ}}$. We always implicitly assume that $\mathcal{S}$ passes all communication between $\mathcal{A}$ and $\mathcal{Z}$.

**Malicious $\mathsf{P_A}$.** $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ and records the values $(a, c)$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$. When $\mathcal{A}$ sends the message $a'$,then $\mathcal{S}$ defined $\beta := a' + a \in \mathbb{F}_p$ and $\delta := c$.

2. $\mathcal{S}$ picks $d \leftarrow \mathbb{F}_{p^r}$ and sends it to $\mathcal{A}$. For each $i \in [h]$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{OT}}$ and receives $\bar{\alpha}_i \in \{0,1\}$ from $\mathcal{A}$, and executes the following:

   (a) If $1 \leq i < h$, sample $K_{\bar{\alpha}_i}^i \leftarrow \{0,1\}^\kappa$. Otherwise $(i = h)$, sample $K_{\bar{\alpha}_h}^h \leftarrow \mathbb{F}_{p^r}$.

   (b) Send $K_{\bar{\alpha}_i}^i$ to $\mathcal{A}$.

   $\mathcal{S}$ defines an index $\alpha = \alpha_1 \cdots \alpha_h \in [0, n)$.

3. Simulator $\mathcal{S}$ defines $\boldsymbol{u} \in \mathbb{F}_p^n$ as a vector that is 0 everywhere except that $\boldsymbol{u}[\alpha] = \beta$, and runs $\mathsf{GGM'}(\alpha, \{K_{\bar{\alpha}_i}^i\}_{i=1}^h)$ to obtain $\{v_i\}_{i \neq \alpha}$. Then, $\mathcal{S}$ sets $\boldsymbol{w}[i] = v_i$ for $i \in [0, n), i \neq \alpha$ and computes $\boldsymbol{w}[\alpha] := \delta - (d + \sum_{i \in [0,n) \setminus \{\alpha\}} \boldsymbol{w}[i])$. Next, $\mathcal{S}$ sends $(\mathsf{sp\text{-}extend}, n)$, $\boldsymbol{u}$ and $\boldsymbol{w}$ to $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$.

4. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ and receives $(\boldsymbol{x}, \boldsymbol{z})$ from $\mathcal{A}$. Then, $\mathcal{S}$ receives $\{\chi_i\}_{i \in [0,n)}$ and $\boldsymbol{x}^*$ from $\mathcal{A}$. $\mathcal{S}$ computes $\boldsymbol{x}' := \boldsymbol{x}^* + \boldsymbol{x} \in \mathbb{F}_p^r$, and defines $x' = \sum_{i=0}^{r-1} \boldsymbol{x}'[i] \cdot g^i$.

   Note that if adversary $\mathcal{A}$ behaves semi-honestly, then $x' = \beta \cdot \left(\sum_{i=0}^{r-1} \boldsymbol{\chi}_\alpha[i] \cdot g^i\right) = \beta \cdot \chi_\alpha \in \mathbb{F}_{p^r}$.

5. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{EQ}}$ and receives an element $V_{\mathsf{A}}' \in \mathbb{F}_{p^r}$ from $\mathcal{A}$. Then $\mathcal{S}$ computes $V_{\mathsf{A}} := \sum_{i \in [0,n)} \chi_i \cdot \boldsymbol{w}[i] - \sum_{i=0}^{r-1} \boldsymbol{z}[i] \cdot g^i \in \mathbb{F}_{p^r}$ and does the following:

   - If $x' = \beta \cdot \chi_\alpha$, then $\mathcal{S}$ checks whether $V_{\mathsf{A}} = V_{\mathsf{A}}'$. If so, $\mathcal{S}$ sends $\mathsf{true}$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ sends $\mathsf{abort}$ to $\mathcal{A}$ and aborts.
   - Otherwise, $\mathcal{S}$ computes $\Delta' := (V_{\mathsf{A}} - V_{\mathsf{A}}') / (\beta \cdot \chi_\alpha - x') \in \mathbb{F}_{p^r}$ and make a global key query $(\Delta')$ to functionality $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$. If $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$ returns $\mathsf{success}$, $\mathcal{S}$ sends $\mathsf{true}$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ sends $\mathsf{abort}$ to $\mathcal{A}$ and aborts.

Firstly, we note that the extraction of $\alpha$ and $\beta$ is perfect. While $d = \gamma - \sum_{i \in [0,n)} \boldsymbol{v}[i]$ in the real protocol execution, $d \in \mathbb{F}_{p^r}$ is sampled at random by $\mathcal{S}$ in the ideal world execution. Following the previous work [KPTZ13, BW13, BGI14], we easily prove that $\boldsymbol{v}[\alpha]$ is computationally indistinguishable from a random value under the assumption that $G$ and $G'$ are pseudorandom generators. Therefore, it is computationally infeasible to find the difference of $d$. In the real protocol execution, the OT message $K_{\bar{\alpha}_i}^i$ for each $i \in [h]$ is the sum of all the nodes at the either left-hand side $(\bar{\alpha}_i = 0)$ or right-hand side $(\bar{\alpha}_i = 1)$ in the $i$-level of the GGM tree. Nevertheless, $K_{\bar{\alpha}_i}^i$ for each $i \in [h]$ is sampled at random by $\mathcal{S}$ in the ideal world execution. For each $i \in [h]$, adversary $\mathcal{A}$ never obtains the PRG seed on node $\alpha_1 \cdots \alpha_i$. In other words, the PRG seed $s_{\alpha_i^*}^i$ (where $\alpha_i^* := \alpha_1 \cdots \alpha_{i-1} \bar{\alpha}_i$) is computationally indistinguishable from a random value, as $G$ and $G'$ are both pseudorandom generators. Therefore, we have that $K_{\bar{\alpha}_i}^i$ for all $i \in [h]$ generated in the real protocol execution are computationally indistinguishable from random values. According to the GGM tree construction, we easily bound the probability of environment $\mathcal{Z}$, which distinguishes element $d$ and keys $\{K_{\bar{\alpha}_i}^i\}_{i \in [h]}$ between the real world and the ideal world, by $\mathsf{negl}(\kappa)$ based on a standard hybrid argument.

In the real protocol execution, $\mathcal{A}$ may send $\boldsymbol{x}^* := \boldsymbol{x}' - \boldsymbol{x}$ for some vector $\boldsymbol{x}' \in \mathbb{F}_p^r$ to $\mathsf{P_B}$, where $\boldsymbol{x}' = \beta \cdot \boldsymbol{\chi}_\alpha$ if $\mathcal{A}$ behaves semi-honestly. Then $\mathsf{P_B}$ will compute $\boldsymbol{y} := \boldsymbol{y}^* - \Delta \cdot \boldsymbol{x}^* = \boldsymbol{z} - \Delta \cdot \boldsymbol{x}'$. Thus,

$$Y = \sum_{i=0}^{r-1} \boldsymbol{y}[i] \cdot g^i = \sum_{i=0}^{r-1} \boldsymbol{z}[i] \cdot g^i - \Delta \cdot (\sum_{i=0}^{r-1} \boldsymbol{x}'[i] \cdot g^i) = Z - \Delta \cdot x'.$$

Further, we have that

$$V_\mathsf{A} - V_\mathsf{B} = \sum_{i \in [0,n)} \chi_i \cdot (\boldsymbol{w}[i] - \boldsymbol{v}[i]) - (Z - Y) = \Delta \cdot (\beta \cdot \chi_\alpha - x').$$

If $x' = \beta \cdot \chi_\alpha$, then we obtain that $V_\mathsf{A} = V_\mathsf{B}$, and thus $\mathcal{S}$ can use $V_\mathsf{A}$ to check the correctness of $V'_\mathsf{A}$ sent by $\mathcal{A}$ to $\mathcal{F}_\mathsf{EQ}$. Otherwise, $V'_\mathsf{A} = V_\mathsf{A} - \Delta \cdot (\beta \cdot \chi_\alpha - x')$ if and only if $\Delta' = (V_\mathsf{A} - V'_\mathsf{A}) / (\beta \cdot \chi_\alpha - x') = \Delta$. Therefore, $\mathcal{S}$ can use the global key query from $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$ to respond the query sent by $\mathcal{A}$ to $\mathcal{F}_\mathsf{EQ}$.

Below, we analyze the outputs of two parties in the real protocol execution. From $c = b + \Delta \cdot a$, $b = \gamma + \Delta \cdot a'$ and $a = \beta - a'$, we have that the following holds:

$$\delta = c = b + \Delta \cdot a = \gamma + \Delta \cdot a' + \Delta \cdot (\beta - a') = \gamma + \Delta \cdot \beta.$$

Following the proof of correctness in [BCG+19a, Theorem 7], we have that $\boldsymbol{w}[i] = \boldsymbol{v}[i]$ for $i \in [n], i \neq \alpha$. From $d = \gamma - \sum_{i \in [0,n)} \boldsymbol{v}[i]$, we have that:

$$\begin{aligned}
\boldsymbol{w}[\alpha] &= \delta - \Big(d + \sum_{i \in [0,n) \setminus \{\alpha\}} \boldsymbol{w}[i]\Big) \\
&= \gamma + \Delta \cdot \beta - \gamma + \Big(\sum_{i \in [0,n)} \boldsymbol{v}[i] - \sum_{i \in [0,n) \setminus \{\alpha\}} \boldsymbol{w}[i]\Big) \\
&= \Delta \cdot \beta + \boldsymbol{v}[\alpha].
\end{aligned}$$

Therefore, $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$ where $\boldsymbol{u}$ is a vector that is 0 everywhere except that $\boldsymbol{u}[\alpha] = \beta$. That is, if $\mathsf{P_B}$ does not abort in the real protocol execution, it will output a vector $\boldsymbol{v}$ which has the same distribution as the one in the ideal world.

Overall, environment $\mathcal{Z}$ cannot distinguish the real world execution from the ideal world execution, except with probability $\mathsf{negl}(\kappa)$.

**Malicious $\mathsf{P_B}$.** $\mathcal{S}$ has access to $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$, and interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, and records all values from $\mathcal{A}$, including $\Delta \in \mathbb{F}_{p^r}$ and $b \in \mathbb{F}_{p^r}$. On behalf of honest $\mathsf{P_A}$, $\mathcal{S}$ samples $a' \leftarrow \mathbb{F}_p$ and sends it to $\mathcal{A}$. Then $\mathcal{S}$ computes $\gamma := b - \Delta \cdot a'$.

2. $\mathcal{S}$ receives an element $d \in \mathbb{F}_{p^r}$ from $\mathcal{A}$. When playing the role of $\mathcal{F}_\mathsf{OT}$, $\mathcal{S}$ receives the OT messages $\{(K_0^i, K_1^i)\}_{i \in [h]}$ from $\mathcal{A}$. Then $\mathcal{S}$ samples $\beta \leftarrow \mathbb{F}_p^*$ and computes $\delta := \gamma + \Delta \cdot \beta$. Given $\{(K_0^i, K_1^i)\}_{i \in [h]}$ and $(d, \delta)$, $\mathcal{S}$ computes a vector $\boldsymbol{w}_\alpha$ for each $\alpha \in [0, n)$ as follows:

   (a) Run $\mathsf{GGM'}(\alpha, \{K_{\bar{\alpha}_i}^i\}_{i=1}^h)$ to obtain $\{v_i(\alpha)\}_{i \neq \alpha}$. Set $\boldsymbol{w}_\alpha[i] = v_i(\alpha)$ for $i \in [0, n), i \neq \alpha$.

   (b) Compute $\boldsymbol{w}_\alpha[\alpha] := \delta - (d + \sum_{i \in [0,n) \setminus \{\alpha\}} \boldsymbol{w}_\alpha[i])$.

3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ and records $\boldsymbol{y}^*$ received from adversary $\mathcal{A}$. Then, $\mathcal{S}$ samples $\chi_0, \ldots, \chi_{n-1} \leftarrow \mathbb{F}_{p^r}$ and $\boldsymbol{x}^* \leftarrow \mathbb{F}_p^r$, and sends them to $\mathcal{A}$. Following the protocol specification, $\mathcal{S}$ computes $\boldsymbol{y} := \boldsymbol{y}^* - \Delta \cdot \boldsymbol{x}^*$ and $Y := \sum_{i=0}^{r-1} \boldsymbol{y}[i] \cdot g^i \in \mathbb{F}_{p^r}$.

4. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{EQ}}$ and receives an element $V_{\mathsf{B}} \in \mathbb{F}_{p^r}$ from adversary $\mathcal{A}$. Then $\mathcal{S}$ computes a set $I$ corresponding to the adversary's guess on the index $\alpha^\star$ of honest $\mathsf{P_A}$ as follows:

   (a) For each $\alpha \in [0, n)$, compute $V_{\mathsf{A}}(\alpha) := \sum_{i\in[0,n)} \chi_i \cdot \boldsymbol{w}_\alpha[i] - \Delta \cdot \beta \cdot \chi_\alpha - Y$.

   (b) Define the set $I = \{\alpha \in [0, n) \mid V_{\mathsf{A}}(\alpha) = V_{\mathsf{B}}\} \subseteq [0, n)$. Note that if adversary $\mathcal{A}$ behaved semi-honestly, then we should have that $V_{\mathsf{A}}(0) = V_{\mathsf{A}}(1) = \cdots = V_{\mathsf{A}}(n-1) = V_{\mathsf{B}}$.

   If $I = \emptyset$, $\mathcal{S}$ picks $\hat{\alpha} \leftarrow [0, n)$ and sends $(\mathsf{false}, W_{\hat{\alpha}})$ to $\mathcal{A}$, and then aborts.

5. Given the OT messages $\{(K_0^i, K_1^i)\}_{i\in[h]}$ and element $d$, $\mathcal{S}$ chooses any $\alpha \in I$ and computes a vector $\boldsymbol{v}$ as the output of $\mathsf{P_B}$ as follows:

   (a) Run $\mathsf{GGM'}(\alpha, \{K^i_{\bar{\alpha}_i}\}_{i=1}^h)$ to obtain $\{v_i(\alpha)\}_{i\neq\alpha}$. Then set $\boldsymbol{v}_\alpha[i] := v_i(\alpha)$ for $i \in [0, n), i \neq \alpha$.

   (b) Compute $\boldsymbol{v}_\alpha[\alpha] = K^h_{\alpha_h} + \sum_{j\in[0,2^{h-1}),j\neq\alpha_1\cdots\alpha_{h-1}} v_{2j+\alpha_h}(\alpha)$.

   (c) Compute an adversarially chosen error $E_\alpha := \gamma - d - \sum_{i\in[0,n)} \boldsymbol{v}_\alpha[i]$.

   (d) For $i \in [0, n)$, define $\boldsymbol{v}[i] = \boldsymbol{v}_\alpha[i]$ if $i \neq \alpha$ and $\boldsymbol{v}[\alpha] = \boldsymbol{v}_\alpha[\alpha] + E_\alpha$ otherwise.

6. $\mathcal{S}$ sends $(\mathsf{sp\text{-}extend}, n)$ and $\boldsymbol{v}$ to $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$. Then, $\mathcal{S}$ sends the set $I$ to $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$. If receiving $\mathsf{success}$ from this functionality, $\mathcal{S}$ sends $(\mathsf{true}, V_{\mathsf{B}})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{EQ}}$. Otherwise, $\mathcal{S}$ picks $\hat{\alpha} \leftarrow [0, n)\backslash I$ and sends $(\mathsf{false}, V_{\mathsf{A}}(\hat{\alpha}))$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{EQ}}$, and then aborts.

In the real protocol execution, $a' \in \mathbb{F}_p$ sent by $\mathsf{P_A}$ is uniformly random, as it is masked by a uniform element $a$ output by $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$. Thus, the element $a'$ simulated by $\mathcal{S}$ has the same distribution as the real value. Note that the actual output $\beta^\star$ of honest $\mathsf{P_A}$ is unknown for $\mathcal{S}$. Simulator $\mathcal{S}$ only uses a dummy element $\beta \in \mathbb{F}_p^*$ sampled uniformly by itself to extract a guess set $I$, and never uses it elsewhere. Environment $\mathcal{Z}$ cannot notice the difference of computing the set $I$ between a real value $\beta^\star$ and a dummy element $\beta$, as $\beta$ has the same distribution as $\beta^\star$. In the real protocol execution, $\boldsymbol{x}^* \in \mathbb{F}_p^r$ is masked by a random vector $\boldsymbol{x} \in \mathbb{F}_p^r$ and is uniformly distributed in $\mathbb{F}_p^r$. Therefore, the vector $\boldsymbol{x}^*$ simulated by $\mathcal{S}$ has the same distribution as the real vector.

The set $I$ extracted by $\mathcal{S}$ corresponds to the selective failure attack on the index $\alpha^\star$ of honest $\mathsf{P_A}$ mounted by $\mathcal{A}$. If $\mathcal{S}$ receives $\mathsf{abort}$ from $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$ (or $\mathcal{S}$ aborts for $I = \emptyset$), we have that $\alpha^\star \notin I$. In the real protocol execution, if $V_{\mathsf{B}} \neq V_{\mathsf{A}}(\alpha^\star)$, the honest $\mathsf{P_A}$ aborts. By previous considerations, this is equivalent to $\alpha^\star \notin I$. Therefore, $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$ aborts if and only if the real protocol execution aborts. In the case of honest $\mathsf{P_A}$, the index $\alpha^\star \in [0, n)$ is sampled uniformly at random in both the real world execution and the ideal world execution. If receiving $\mathsf{abort}$ from $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$, then $\mathcal{S}$ needs to send $\mathsf{false}$ along with an element $V_{\mathsf{A}}(\hat{\alpha}) \neq V_{\mathsf{B}}$ to adversary $\mathcal{A}$. Although $\mathcal{S}$ does not know the actual index $\alpha^\star$, it can sample a random index $\hat{\alpha}$ from the set $[0, n)\backslash I$ and send the element $V_{\mathsf{A}}(\hat{\alpha})$ to $\mathcal{A}$. In the case of aborting, this simulation is perfect, since $\mathcal{Z}$ cannot obtain the output of $\mathsf{P_A}$ due to aborting, and the dummy index $\hat{\alpha}$ has the same distribution as the actual index $\alpha^\star$ under the condition that $I$ is an incorrect guess.

Below, we prove that except with probability $1/p^r$, the output vector $\boldsymbol{v} \in \mathbb{F}_{p^r}^n$ computed by $\mathcal{S}$ is independent from the choice $\alpha \in I$, and is correct such that $\boldsymbol{w}^\star = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}^\star$ if the protocol execution does not abort, where $\boldsymbol{u}^\star, \boldsymbol{w}^\star$ are the output of honest $\mathsf{P_A}$ and $\boldsymbol{u}^\star$ is a vector with exactly one nonzero entry $\boldsymbol{u}^\star[\alpha^\star] = \beta^\star$.

**Claim 1.** *Except with probability at most $1/p^r$, all choices of $\alpha, \alpha' \in I$ in the step 5 of the simulation lead to the same output vector $\boldsymbol{v}$.*

*Proof.* For the case of $|I| = 1$, this is trivial. For $|I| > 1$, we prove that for any two indices $\alpha, \alpha' \in I$, $\boldsymbol{v}_\alpha[i] = \boldsymbol{v}_{\alpha'}[i]$ for $i \in [0, n), i \neq \alpha, \alpha'$. In particular, from $V_{\mathsf{A}}(\alpha) = V_{\mathsf{A}}(\alpha') = V_{\mathsf{B}}$, we have that:

$$\begin{aligned}
&\sum_{i\in[0,n)} \chi_i \cdot \boldsymbol{w}_\alpha[i] - \Delta \cdot \beta \cdot \chi_\alpha - Y = \sum_{i\in[0,n)} \chi_i \cdot \boldsymbol{w}_{\alpha'}[i] - \Delta \cdot \beta \cdot \chi_{\alpha'} - Y \\
\Leftrightarrow &\sum_{i\in[0,n)\backslash\{\alpha,\alpha'\}} \chi_i \cdot (\boldsymbol{w}_\alpha[i] - \boldsymbol{w}_{\alpha'}[i]) + \chi_\alpha \cdot (\boldsymbol{w}_\alpha[\alpha] - \boldsymbol{w}_{\alpha'}[\alpha] - \Delta \cdot \beta) \\
&+ \chi_{\alpha'} \cdot (\boldsymbol{w}_\alpha[\alpha'] - \boldsymbol{w}_{\alpha'}[\alpha'] + \Delta \cdot \beta) = 0.
\end{aligned}$$

Note that $\Delta$, $\beta$, $\boldsymbol{w}_\alpha$ and $\boldsymbol{w}_{\alpha'}$ have already been defined before $\{\chi_i\}_{i\in[0,n)}$ is sampled. Furthermore, each coefficient $\chi_i$ is uniformly random. Therefore, except with probability $1/p^r$, we have that:

$$\begin{aligned}
&\boldsymbol{w}_\alpha[i] = \boldsymbol{w}_{\alpha'}[i] \text{ for } i \in [0, n)\backslash\{\alpha, \alpha'\}, \\
&\boldsymbol{w}_\alpha[\alpha] - \boldsymbol{w}_{\alpha'}[\alpha] = \boldsymbol{w}_{\alpha'}[\alpha'] - \boldsymbol{w}_\alpha[\alpha'] = \Delta \cdot \beta.
\end{aligned}$$

For all $\alpha \in [n]$, we have that $\boldsymbol{v}_\alpha[i] = \boldsymbol{w}_\alpha[i]$ for $i \in [0, n), i \neq \alpha$ from their definitions. Thus, we obtain that $\boldsymbol{v}_\alpha[i] = \boldsymbol{v}_{\alpha'}[i]$ for all $i \in [0, n), i \neq \alpha, \alpha'$. From the equations $\boldsymbol{w}_\alpha[\alpha] = \gamma + \Delta \cdot \beta - (d + \sum_{i\in[0,n)\backslash\{\alpha\}} \boldsymbol{w}_\alpha[i])$ and $E_\alpha = \gamma - d - \sum_{i\in[0,n)} \boldsymbol{v}_\alpha[i]$, we also have that $\boldsymbol{w}_\alpha[\alpha] = (\boldsymbol{v}_\alpha[\alpha] + E_\alpha) + \Delta \cdot \beta$. According to $\boldsymbol{w}_\alpha[\alpha] - \boldsymbol{v}_{\alpha'}[\alpha] = \Delta \cdot \beta$, we further have that $\boldsymbol{v}_{\alpha'}[\alpha] = \boldsymbol{v}_\alpha[\alpha] + E_\alpha$. Overall, for all $\alpha, \alpha' \in I$, $\mathcal{S}$ would compute the same output vector $\boldsymbol{v}$, which completes the proof. □

It is obvious that $\boldsymbol{w}^\star = \boldsymbol{v} + \boldsymbol{u}^\star \cdot \Delta$ in the ideal world execution, where recall that $\boldsymbol{u}^\star, \boldsymbol{w}^\star$ are the output of honest $\mathsf{P}_{\mathsf{A}}$. In the following, we focus on proving that this equation still holds in the real protocol execution, except with probability $1/p^r$. We define a vector $\boldsymbol{v}^\star$ as $\boldsymbol{v}^\star[i] = \boldsymbol{v}_{\alpha^\star}[i]$ for $i \neq \alpha^\star$ and $\boldsymbol{v}^\star[\alpha^\star] = \boldsymbol{v}_{\alpha^\star}[\alpha^\star] + E_{\alpha^\star}$, where $E_{\alpha^\star} = \gamma - d - \sum_{i\in[0,n)} \boldsymbol{v}_{\alpha^\star}[i]$. Based on the analysis in Claim 1, we have that $\boldsymbol{w}^\star[i] = \boldsymbol{v}^\star[i]$ for all $i \neq \alpha^\star$ and $\boldsymbol{w}^\star[\alpha^\star] = (\boldsymbol{v}_{\alpha^\star}[\alpha^\star] + E_{\alpha^\star}) + \Delta \cdot \beta^\star = \boldsymbol{v}^\star[\alpha^\star] + \Delta \cdot \beta^\star$. Therefore, we obtain that $\boldsymbol{w}^\star = \boldsymbol{v}^\star + \Delta \cdot \boldsymbol{u}^\star$. In the real protocol execution, adversary $\mathcal{A}$ can compute an output vector $\boldsymbol{v}$ following the approach used by simulator $\mathcal{S}$ (i.e., the step 5 of the simulation). Based on the Claim 1, if $\alpha^\star \in I$ (i.e., the real protocol execution does not abort), we have that $\boldsymbol{v} = \boldsymbol{v}^\star$ except with probability $1/p^r$, and thus $\boldsymbol{w}^\star = \boldsymbol{v} + \boldsymbol{u}^\star \cdot \Delta$. In conclusion, any environment $\mathcal{Z}$ cannot distinguish the real world execution from the ideal world execution, except with probability $1/p^r$. □

**Implementation and optimizations.** We discuss some implementation details and optimizations for the protocol in Figure 5.

1. When $p \geq 2^\rho$, the parties can use the output of $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ directly as $[\beta]$ in step 2, since the value $\beta$ will be nonzero with overwhelming probability.

2. In the consistency check procedure, $\mathsf{P}_{\mathsf{A}}$ can send a uniform $\mathsf{seed} \in \{0,1\}^\kappa$ to $\mathsf{P}_{\mathsf{B}}$, and then both parties can simply derive the $\{\chi_i\}$ from $\mathsf{seed}$ using a hash function modeled as a random oracle.

3. When $t$ extend executions of spsVOLE are needed, we can perform the consistency check in a batch. This optimization reduces the need of $t \cdot r$ sVOLE correlations to only $r$ sVOLE correlations. Specifically, based on the batched idea by Kang et al. [YWL+20], we can combine $t$ consistency checks into only one check as follows:

   (a) After $t$ sVOLE correlations $((\boldsymbol{u}_j, \boldsymbol{w}_j), \boldsymbol{v}_j)$ for $j \in [t]$ have been computed, where $\boldsymbol{u}_j$ is a vector that is 0 everywhere except that $\boldsymbol{u}_j[\alpha_j] = \beta_j$, $\mathsf{P}_{\mathsf{A}}$ and $\mathsf{P}_{\mathsf{B}}$ send $(\mathsf{extend}, r)$ to $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, which returns $(\boldsymbol{x}, \boldsymbol{z}) \in \mathbb{F}_p^r \times \mathbb{F}_{p^r}^r$ to $\mathsf{P}_{\mathsf{A}}$ and $\boldsymbol{y} \in \mathbb{F}_{p^r}^r$ to $\mathsf{P}_{\mathsf{B}}$.

(b) $\mathsf{P_A}$ samples $\chi_{i,j} \leftarrow \mathbb{F}_{p^r}$ for $i \in [0,n), j \in [t]$, and writes $\chi_{\alpha_j} = \sum_{i=0}^{r-1} \boldsymbol{\chi}_j[i] \cdot g^i$. It then computes $\boldsymbol{x}^* := \boldsymbol{x} - \sum_{j\in[t]} \beta_j \cdot \boldsymbol{\chi}_j \in \mathbb{F}_p^r$ and sends $\{\chi_{i,j}\}_{i\in[0,n),j\in[t]}$ and $\boldsymbol{x}^*$ to $\mathsf{P_B}$.

(c) $\mathsf{P_B}$ computes $\boldsymbol{y}^* := \boldsymbol{y} + \Delta \cdot \boldsymbol{x}^* \in \mathbb{F}_{p^r}^r$, $Y := \sum_{i=0}^{r-1} \boldsymbol{y}^*[i] \cdot g^i \in \mathbb{F}_{p^r}$, and $V_{\mathsf{B}} := \sum_{i\in[0,n)} \sum_{j\in[t]} \chi_{i,j} \cdot \boldsymbol{v}_j[i] - Y \in \mathbb{F}_{p^r}$. $\mathsf{P_A}$ computes $Z := \sum_{i=0}^{r-1} \boldsymbol{z}[i] \cdot g^i \in \mathbb{F}_{p^r}$ and $V_{\mathsf{A}} := \sum_{i\in[0,n)} \sum_{j\in[t]} \chi_{i,j} \cdot \boldsymbol{w}_j[i] - Z \in \mathbb{F}_{p^r}$. Then $\mathsf{P_A}$ sends $V_{\mathsf{A}}$ to $\mathcal{F}_{\mathsf{EQ}}$, and $\mathsf{P_B}$ sends $V_{\mathsf{B}}$ to $\mathcal{F}_{\mathsf{EQ}}$.

(d) If either party receives $\mathsf{false}$ or $\mathsf{abort}$ from $\mathcal{F}_{\mathsf{EQ}}$, it aborts.

Following the analysis [YWL+20], the above batched consistency check guarantees the correctness of outputs of two parties and the consistency of $\Delta$ among $t$ extend executions. We note that the batched consistency check *does not* reveal more information on indices $\{\alpha_j\}_{j\in[t]}$ than executing $t$ independent consistency checks described in Figure 5.

## 4.2 A More Efficient sVOLE Protocol

**Overview of our protocol.** We use an LPN variant which states that uniform $(\mathbf{A}, \boldsymbol{x}) \in \mathbb{F}_p^{k\times n} \times \mathbb{F}_p^n$ is computationally indistinguishable from $(\mathbf{A}, \boldsymbol{s}\cdot\mathbf{A} + \boldsymbol{e})$, where $\boldsymbol{s} \in \mathbb{F}_p^k$ is uniform and $\boldsymbol{e} \in \mathbb{F}_p^n$ has a small, fixed Hamming weight $t$. To increase efficiency, we consider a *regular* noise distribution, which has been used in previous work [HOSS18, BCG+19a, YWL+20]. Specifically, the error vector $\boldsymbol{e} \in \mathbb{F}_p^n$ is divided into $t$ consecutive sub-vectors of length $\lfloor n/t \rfloor$, where each sub-vector has exactly one uniform non-zero entry. Recall that our goal is to generate a vector of authenticated values (i.e., sVOLE) $[\boldsymbol{x}]$. Using the linearity of LPN, two parties just need to generate authenticated values $[\boldsymbol{s}]$ and $[\boldsymbol{e}]$. We can generate $[\boldsymbol{e}]$ by computing spsVOLE $t$ times with each of length $\lfloor n/t \rfloor$ and then concatenating them together. The communication cost to generate $[\boldsymbol{e}]$ is $O(t \log \frac{n}{t})$. Then the LPN assumption essentially provides us a way to generate $n$ authenticated values using the communication cost to compute $k$ authenticated values along with $O(t \log \frac{n}{t})$. Based on the iteration idea [YWL+20], we can first generate $k$ authenticated values using a one-time setup protocol, and then output $n-k$ authenticated values for each iteration using only $O(t \log \frac{n}{t})$ cost and remain $k$ authenticated values for the next iteration. In this case, the average communication cost for each authenticated value is about $O(t \log \frac{n}{t})/(n-k)$. Since $n \gg t$ and $n \gg k$, the protocol is very efficient.

### 4.2.1 An LPN Variant

Our sVOLE protocol depends on a variant of the Learning Parity with Noise (LPN) assumption [BFKL94], called *LPN with static leakage* by Boyle et al. [BCG+19a]. Here, the adversary is allowed to query for a single bit of leakage (on average) about the error vector. Following prior works [HOSS18, BCG+19a], we consider a regular error distribution with fixed Hamming weight.

**Definition 1** (LPN with static leakage [BCG+19a])**.** *Let $\mathcal{G}$ be a polynomial-time code generation algorithm such that $\mathcal{G}(k,n,\mathbb{F}_p)$ outputs a matrix $\mathbf{A} \in \mathbb{F}_p^{k\times n}$, and $\mathcal{D}_{n,t}$ be a regular error distribution over $\mathbb{F}_p^n$ with fixed Hamming weight $t$ . Parameters $k, n, t$ are functions of $\kappa$. Consider the following game $G_b(1^\kappa)$ involving a (PPT) adversary $\mathcal{A}$ parameterized by $\kappa$:*

1. *Sample $\mathbf{A} \leftarrow \mathcal{G}(k,n,\mathbb{F}_p)$ and $\boldsymbol{u} \leftarrow \mathbb{F}_p^k$. Also sample an error vector $\boldsymbol{e} \leftarrow \mathcal{D}_{n,t}$. Let $\alpha_1, \ldots, \alpha_t$ be the indices of the nonzero entries in $\boldsymbol{e}$, each of which is located in a disjoint $\lfloor n/t \rfloor$-sized interval.*

2. Selective failure query: *$\mathcal{A}$ sends $t$ sets $I_1, \ldots, I_t \subseteq [0,n)$. If $\alpha_i \in I_i$ for all $i \in [t]$, then send $\mathsf{success}$ to $\mathcal{A}$; otherwise, abort and define the output of $\mathcal{A}$ as 0.*

**Protocol $\Pi^{p,r}_{\mathsf{sVOLE}}$**

**Parameters:** LPN parameters $n, k, t, \ell, m$ with $k, t < n$, $\ell = n - k$, and $m = n/t$. A matrix $\mathbf{A} \in \mathbb{F}_p^{k \times n}$ output by a code generator $\mathcal{G}(k, n, \mathbb{F}_p)$.

**Initialize:** This procedure is executed only once.

1. $\mathsf{P_A}$ and $\mathsf{P_B}$ send $\mathsf{init}$ to $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$, which returns $\Delta \in \mathbb{F}_{p^r}$ to $\mathsf{P_B}$.
2. $\mathsf{P_A}$ and $\mathsf{P_B}$ send $(\mathsf{extend}, k)$ to $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$, which returns $(\boldsymbol{u}, \boldsymbol{w}) \in \mathbb{F}_p^k \times \mathbb{F}_{p^r}^k$ to $\mathsf{P_A}$ and $\boldsymbol{v} \in \mathbb{F}_{p^r}^k$ to $\mathsf{P_B}$ such that $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$.

**Extend:** This procedure can be called multiple times, and produces $\ell$ sVOLE correlations in each iteration.

3. For $i \in [t]$, $\mathsf{P_A}$ and $\mathsf{P_B}$ send $(\mathsf{sp\text{-}extend}, m)$ to $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$, which returns $(\boldsymbol{a}_i, \boldsymbol{c}_i) \in \mathbb{F}_p^m \times \mathbb{F}_{p^r}^m$ to $\mathsf{P_A}$ and $\boldsymbol{b}_i \in \mathbb{F}_{p^r}^m$ to $\mathsf{P_B}$ such that $\boldsymbol{c}_i = \boldsymbol{b}_i + \Delta \cdot \boldsymbol{a}_i$ and $\boldsymbol{a}_i \in \mathbb{F}_p^m$ has exactly one nonzero entry.

   In any of these $\mathsf{spsVOLE}$ executions, if either party receives $\mathsf{abort}$ from $\mathcal{F}^{p,r}_{\mathsf{spsVOLE}}$, it aborts.
4. $\mathsf{P_A}$ defines $\boldsymbol{e} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_t) \in \mathbb{F}_p^n$ and a vector $\boldsymbol{t} = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_t) \in \mathbb{F}_{p^r}^n$. Then $\mathsf{P_A}$ computes $\boldsymbol{x} := \boldsymbol{u} \cdot \mathbf{A} + \boldsymbol{e} \in \mathbb{F}_p^n$ and $\boldsymbol{z} := \boldsymbol{w} \cdot \mathbf{A} + \boldsymbol{t} \in \mathbb{F}_{p^r}^n$.

   $\mathsf{P_B}$ defines $\boldsymbol{s} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_t) \in \mathbb{F}_{p^r}^n$ and computes $\boldsymbol{y} := \boldsymbol{v} \cdot \mathbf{A} + \boldsymbol{s} \in \mathbb{F}_{p^r}^n$.
5. $\mathsf{P_A}$ updates $\boldsymbol{u}, \boldsymbol{w}$ by setting $\boldsymbol{u} := \boldsymbol{x}[0 : k) \in \mathbb{F}_p^k$ and $\boldsymbol{w} := \boldsymbol{z}[0 : k) \in \mathbb{F}_{p^r}^k$, and outputs $(\boldsymbol{b}, \mathsf{M}[\boldsymbol{b}]) := (\boldsymbol{x}[k : n), \boldsymbol{z}[k : n)) \in \mathbb{F}_p^\ell \times \mathbb{F}_{p^r}^\ell$.

   $\mathsf{P_B}$ updates $\boldsymbol{y}$ by setting $\boldsymbol{v} := \boldsymbol{y}[0 : k) \in \mathbb{F}_{p^r}^k$, and outputs $\mathsf{K}[\boldsymbol{b}] := \boldsymbol{y}[k : n) \in \mathbb{F}_{p^r}^\ell$.

Figure 6: Iterative sVOLE protocol in the $(\mathcal{F}^{p,r}_{\mathsf{sVOLE}}, \mathcal{F}^{p,r}_{\mathsf{spsVOLE}})$-hybrid model.

*3. If $b = 1$, let $\boldsymbol{x} = \boldsymbol{u} \cdot \mathbf{A} + \boldsymbol{e}$; otherwise, sample $\boldsymbol{x} \leftarrow \mathbb{F}_p^n$.*

*4. Send $\boldsymbol{x}$ to $\mathcal{A}$, who then outputs a bit $b'$.*

*The $(\mathcal{D}_{n,t}, \mathcal{G}, \mathbb{F}_p)$-$\mathsf{LPN}(k, n, t)$ assumption with static leakage states that*

$$\left| \Pr\left[\mathcal{A}^{G_0(1^\kappa)} = 1\right] - \Pr\left[\mathcal{A}^{G_1(1^\kappa)} = 1\right] \right| \leq \mathsf{negl}(\kappa).$$

For our applications, we will choose the code matrix $\mathbf{A}$ from a family of *d-local linear codes*, i.e., let $\mathcal{G}$ generate a matrix in which each column is uniform subject to having exactly $d$ number of nonzero entries, where $d$ is a small constant. This is advantageous since such matrices support efficient matrix-vector multiplications (*linear* in $n$). Note that the hardness of LPN for local linear codes is a well-established assumption [Ale03]. Following prior works [HOSS18, BCG+19a], we will use the following error distribution $\mathcal{D}_{n,t}$: the uniform, weight $t$ vectors over $\mathbb{F}_p^n$ in a *regular* form. That is, we consider the case where the error vector $\boldsymbol{e} \in \mathbb{F}_p^n$ is divided into $t$ consecutive sub-vectors of length $\lfloor n/t \rfloor$, with each sub-vector uniform subject to having exactly one nonzero entry.

### 4.2.2 Construction

We present the sVOLE protocol with iterations in Figure 6, which works in the $(\mathcal{F}^{p,r}_{\mathsf{sVOLE}}, \mathcal{F}^{p,r}_{\mathsf{spsVOLE}})$-hybrid model. In the one-time setup phase, $\mathsf{P_A}$ can use $\mathcal{G}(k, n, \mathbb{F}_p)$ to output a matrix $\mathbf{A} \in \mathbb{F}_p^{k \times n}$, and sends it to $\mathsf{P_B}$. This communication can be further reduced by sampling a random seed from $\{0,1\}^\kappa$ and computing $\mathbf{A}$ with the seed and $\mathcal{G}(k, n, \mathbb{F}_p)$ in the random oracle model.

In our sVOLE protocol, we assume that $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$ shares the same initialization procedure with $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ (adopting the same global key $\Delta$). Recall that protocol $\Pi_{\mathsf{spsVOLE}}^{p,r}$ shown in Figure 5 works in the $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$-hybrid model, and adopts the initialization procedure in $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ to setup the global key. In other words, if we use the protocol $\Pi_{\mathsf{spsVOLE}}^{p,r}$ to securely compute the functionality $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$, then this assumption holds.

For an honest execution of protocol $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, we show that the outputs of two parties satisfy the correct correlation. Since $\boldsymbol{c}_i = \boldsymbol{b}_i + \Delta \cdot \boldsymbol{a}_i$ for $i \in [t]$ from the spsVOLE definition, we have $\boldsymbol{t} = \boldsymbol{s} + \Delta \cdot \boldsymbol{e}$. Together with $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$ from the definition of $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, we obtain that the following holds for the first iteration:

$$\boldsymbol{z} - \boldsymbol{y} = (\boldsymbol{w} - \boldsymbol{v}) \cdot \mathbf{A} + \boldsymbol{t} - \boldsymbol{s} = \Delta \cdot (\boldsymbol{u} \cdot \mathbf{A} + \boldsymbol{e}) = \Delta \cdot \boldsymbol{x}.$$

In any subsequent iteration, we still have that $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$ holds, as $\boldsymbol{z} = \boldsymbol{y} + \Delta \cdot \boldsymbol{x}$ in the previous iteration. Thus, in all subsequent iterations, we obtain that $\boldsymbol{z} = \boldsymbol{y} + \Delta \cdot \boldsymbol{x}$.

**Theorem 3.** *Protocol $\Pi_{\mathsf{sVOLE}}^{p,r}$ described in Figure 6 UC-realizes $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ with any polynomial number of sVOLE correlations in the $(\mathcal{F}_{\mathsf{sVOLE}}^{p,r}, \mathcal{F}_{\mathsf{spsVOLE}}^{p,r})$-hybrid model under the $(\mathcal{D}_{n,t}, \mathcal{G}, \mathbb{F}_p)$-$\mathsf{LPN}(k,n,t)$ assumption with static leakage. In particular, any PPT environment $\mathcal{Z}$ cannot distinguish the real world execution from the ideal world execution, except with probability at most $\mathsf{negl}(\kappa)$.*

*Proof.* We first consider the case of a malicious $\mathsf{P_A}$ and then consider the case of a malicious $\mathsf{P_B}$. In each case, we construct a PPT simulator $\mathcal{S}$ which runs a PPT adversary $\mathcal{A}$ as a subroutine, and emulates the functionalities $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ and $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$. We always implicitly assume that $\mathcal{S}$ passes all communication between $\mathcal{A}$ and $\mathcal{Z}$.

**Malicious $\mathsf{P_A}$.** In the setup phase, $\mathcal{S}$ receives a matrix $\mathbf{A} \in \mathbb{F}_p^{k \times n}$ from adversary $\mathcal{A}$. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ and receives vectors $(\boldsymbol{u}, \boldsymbol{w}) \in \mathbb{F}_p^k \times \mathbb{F}_{p^r}^k$ from $\mathcal{A}$. For each iteration, $\mathcal{S}$ has access to functionality $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, and interacts with $\mathcal{A}$ as follows:

1. For $i \in [t]$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$, and receives $\boldsymbol{a}_i \in \mathbb{F}_p^m$ with at most one nonzero entry and $\boldsymbol{c}_i \in \mathbb{F}_{p^r}^m$ from $\mathcal{A}$. Then, $\mathcal{S}$ defines a regular error vector $\boldsymbol{e} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_t) \in \mathbb{F}_p^n$, and sets vector $\boldsymbol{t} = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_t) \in \mathbb{F}_{p^r}^n$.

2. For $i \in [t]$, if $\mathcal{A}$ sends a global key query $\Delta_i'$ to $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$, then $\mathcal{S}$ records $\Delta_i'$. If there exists two different indices $i, j \in [t]$ such that $\Delta_i' \neq \Delta_j'$, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ defines $\Delta' = \Delta_i'$ for some $i \in [t]$, and sends $\Delta'$ to functionality $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$. If receiving $\mathsf{abort}$ from $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, $\mathcal{S}$ aborts.

3. $\mathcal{S}$ computes $\boldsymbol{x} := \boldsymbol{u} \cdot \mathbf{A} + \boldsymbol{e} \in \mathbb{F}_p^n$ and $\boldsymbol{z} := \boldsymbol{w} \cdot \mathbf{A} + \boldsymbol{t} \in \mathbb{F}_{p^r}^n$, and then updates $\boldsymbol{u} := \boldsymbol{x}[0:k) \in \mathbb{F}_p^k$ and $\boldsymbol{w} := \boldsymbol{z}[0:k) \in \mathbb{F}_{p^r}^k$ for the next iteration. $\mathcal{S}$ sends $\boldsymbol{x}[k:n) \in \mathbb{F}_p^\ell$ and $\boldsymbol{z}[k:n) \in \mathbb{F}_{p^r}^\ell$ to functionality $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$.

Clearly, the simulation of $\mathcal{S}$ is perfect. In the real protocol execution, we can see that honest $\mathsf{P_B}$ will compute a vector $\boldsymbol{y}$ for each iteration such that $\boldsymbol{z} = \boldsymbol{y} + \Delta \cdot \boldsymbol{x}$ where $(\boldsymbol{x}, \boldsymbol{z})$ is computed by the adversary, based on the correctness analysis. Overall, it is perfectly indistinguishable between the real world execution and the ideal world execution.

**Malicious $\mathsf{P_B}$.** Simulator $\mathcal{S}$ uses $\mathcal{G}(k,n,\mathbb{F}_p)$ to generate a matrix $\mathbf{A} \in \mathbb{F}_p^{k \times n}$ and sets it as a parameter. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, and receives $\Delta \in \mathbb{F}_{p^r}$ and $\boldsymbol{v} \in \mathbb{F}_{p^r}^k$ from adversary $\mathcal{A}$. $\mathcal{S}$ forwards $\Delta$ to functionality $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$. For each iteration, $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$, and receives a vector $\boldsymbol{b}_i \in \mathbb{F}_{p^r}^m$ from $\mathcal{A}$ for $i \in [t]$.

2. For $i \in [t]$, $\mathcal{A}$ may send a set $I_i \subseteq [0, m)$ to $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$ emulated by $\mathcal{S}$. After receiving $t$ sets, $\mathcal{S}$ samples an error vector $\boldsymbol{e} \leftarrow \mathcal{D}_{n,t}$ and defines $\{\alpha_1, \ldots, \alpha_t\}$ to be the sorted indices of non-zero entries of $\boldsymbol{e}$. Then, for $i \in [t]$, $\mathcal{S}$ updates $\alpha_i := \alpha_i \pmod m$ and checks that $\alpha_i \in I_i$. If any check fails, $\mathcal{S}$ aborts.

3. $\mathcal{S}$ sets $\boldsymbol{s} := (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_t) \in \mathbb{F}_{p^r}^n$, computes $\boldsymbol{y} := \boldsymbol{v} \cdot \mathbf{A} + \boldsymbol{s} \in \mathbb{F}_{p^r}^n$, and updates $\boldsymbol{v} := \boldsymbol{y}[0:k) \in \mathbb{F}_{p^r}^k$ for the next iteration. Then, $\mathcal{S}$ sends $\boldsymbol{y}[k:n) \in \mathbb{F}_{p^r}^\ell$ to functionality $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$.

Note that the probability of aborting is the same in both the real world execution and the ideal world execution, as $\mathcal{S}$ samples an error vector $\boldsymbol{e}$ defined just as in the real protocol execution.

It is easy to see that the simulation of $\mathcal{S}$ is perfect. Based on the correctness analysis, we easily obtain that honest $\mathsf{P_A}$ computed vectors $\boldsymbol{x}, \boldsymbol{z}$ for each iteration such that $\boldsymbol{z} = \boldsymbol{y} + \Delta \cdot \boldsymbol{x}$. Below, we focus on proving that the PPT environment $\mathcal{Z}$ cannot distinguish the honest party's output between the real world execution and the ideal world execution under the LPN assumption with static leakage.

The vector $\boldsymbol{z} = \boldsymbol{y} + \Delta \cdot \boldsymbol{x}$ computed by honest $\mathsf{P_A}$ is always determined by $\boldsymbol{x}$, $\boldsymbol{y}$ and $\Delta$ in both two worlds. Therefore, we focus on proving that vector $\boldsymbol{x}$ computed by honest $\mathsf{P_A}$ in each iteration is computationally indistinguishable from a uniform vector in $\mathbb{F}_p^n$ under the $(\mathcal{D}_{n,t}, \mathcal{G}, \mathbb{F}_p)$-$\mathsf{LPN}(k, n, t)$ assumption with static leakage. This proof proceeds via a sequence of games.

- In Game 0, the protocol execution is simulated by $\mathcal{S}$, which has the identical distribution as the real protocol execution.

- In Game $i$ ($i \geq 1$), the vectors $\boldsymbol{x}$ computed by honest $\mathsf{P_A}$ in the first $i$ iterations are replaced with uniform vectors from $\mathbb{F}_p^n$.

- In the final game, we have that the vectors computed by $\mathsf{P_A}$ in all iterations are uniform, and thus the $\mathsf{P_A}$'s output has the same distribution as that in the ideal world execution.

If there exists a PPT adversary $\mathcal{Z}$ which can distinguish Game $i-1$ from Game $i$, then we can construct a PPT algorithm $\mathcal{B}$ who can break the LPN assumption with static leakage. Specifically, given a matrix $\mathbf{A}^* \in \mathbb{F}_p^{k \times n}$, $\mathcal{B}$ sets $\mathbf{A}^*$ as a parameter, and behaves exactly as in Game $i-1$, except for the following differences in the $i$th iteration:

1. For $i \in [t]$, emulate $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$, receive a set $I_i \in [0, m)$ from the adversary, and update each entry $a$ in $I_i$ as $a := a + (i-1) \cdot m$.

2. Send the updated sets $I_1, \ldots, I_t$ to the LPN game. If this game aborts, then abort and output 0. Otherwise, receive $\mathsf{success}$ from this game, and continue the simulation.

3. After receiving a challenge vector $\boldsymbol{x}^*$ from the LPN game, set $\boldsymbol{x}^*$ as the vector $\boldsymbol{x}$ computed by $\mathsf{P_A}$ in the $i$th iteration.

4. If $\mathcal{Z}$ outputs $i-1$, output 1. Otherwise ($\mathcal{Z}$ outputs $i$), output 0.

The probability that the LPN game aborts is the same as the probability that simulator $\mathcal{S}$ aborts, and thus is identical to that of aborting in the real protocol execution. Note that the secret vector $\boldsymbol{u}$ from either the $(i-1)$th iteration or functionality $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ is uniformly random, and is never revealed to $\mathcal{Z}$. If $\boldsymbol{x}^*$ is sampled from a uniform distribution, then $\mathcal{B}$ behaves exactly as in Game $i$. Otherwise, $\mathcal{B}$ behaves exactly as in Game $i-1$. Therefore, the vector $\boldsymbol{x}$ computed by $\mathsf{P_A}$ in each iteration is computationally indistinguishable from a uniform vector under the LPN assumption with static leakage. □

**Complexity analysis.** Now, we analyze the rounds and communication complexity of our sVOLE protocol in the $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$-hybrid model, where $\mathcal{F}_{\mathsf{spsVOLE}}^{p,r}$ is instantiated by the single-point sVOLE protocol with optimizations shown in Section 4.1. In the spsVOLE protocol, we can use an OT extension protocol such as [YWL+20] to implement $\mathcal{F}_{\mathsf{OT}}$, and need $O(\kappa)$ bits of communication for each OT correlation. Note that the OT extension protocol can be executed in parallel with our sVOLE protocol. Therefore, for one iteration, our sVOLE protocol requires 4 rounds for the case $r \log p \geq \kappa$ (resp., 6 rounds for the case $\rho \leq r \log p < \kappa$), and needs a communication of $O(t\kappa \log \frac{n}{t})$ bits.

**Optimizations.** We observe that our single-point sVOLE protocol needs $r+1$ sVOLE correlations, which can be optimized using the iteration idea. Specifically, $n_0 = k + t(r+1)$ sVOLE correlations are needed to execute one iteration of our protocol $\Pi_{\mathsf{sVOLE}}$. Now in the one-time setup phase, we can compute $n_0$ sVOLE correlations, which are sufficient for the first iteration. By remaining $n_0$ sVOLE correlations from this iteration, we can execute the next iteration without calling extra sVOLE correlations. Using the above optimization, we can output $n - n_0$ sVOLE correlations per iteration by consuming $n_0$ sVOLE correlations and communicating $O(t\kappa \log \frac{n}{t})$ bits.

Now we turn our attention to the one-time setup phase, where we need to compute $n_0$ sVOLE correlations. We can again use the iteration idea to optimize the cost. Specifically, we can choose another LPN parameters $(n_0, k_0, t_0)$. We can use the base sVOLE protocol described in Appendix B to first generate about $n_0' = k_0 + t_0(r+1)$ sVOLE correlations by communicating $O(n_0' r \log^2 p)$ bits, and then compute $n_0$ sVOLE correlations by *a single iteration* of protocol $\Pi_{\mathsf{sVOLE}}$ with a communication of $O(t_0 \kappa \log \frac{n_0}{t_0})$ bits.

# 5 Performance Evaluation

## 5.1 Practical Optimizations

**Pipelining.** We use pipelining [HEKM11] to streamline the protocol execution, where we send the messages for some gate while computing for the next gate in the circuit. Pipelining has many advantages: 1) it reduces the execution time, since we perform network transmission and computation at the same time; 2) it reduces the memory usage significantly. Without pipelining, we would need to store all values in memory before sending them out, which takes memory linear to the circuit size. Now we only need memory that is proportional to the memory needed to evaluate the circuit in the clear.

**Batch checking of multiplication triples.** The other memory overhead is the need to store all triples used to check the correctness of the execution (our main protocol step 6 to step 8). In our implementation, we instead make a trade-off between round complexity and memory use. We buffer a set of $m$ gates and perform a batch checking on all of them together. As a result, the round complexity is $C/m$ and the memory usage is about $m\kappa$.

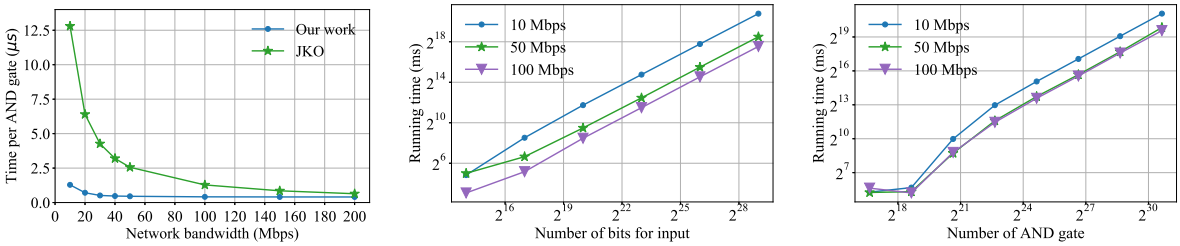
If the batch size is too small, we will incur a higher checking overhead as the bucket size has to be large. To minimize the memory overhead, we find concretely the minimum batch size in the cut-and-choose game. From the proof of Theorem 1, we conclude that if we set $c = B$, we have

$$\binom{BC+B}{B} = \frac{\prod_{i=1}^{B}(BC+i)}{B!} = \prod_{i=0}^{B-1}\left(\frac{B}{B-i}C+1\right) \geq (C+1)^B \geq 2^\rho.$$

Therefore, we obtain that $B \geq \rho/\log(C+1)$. Our goal is to minimize both bucket size $B$ and the circuit size $C$. Practically, we only consider the cases when $B = 2$ and $B = 3$. The Table 2 shows

| | $\rho = 40$ | $\rho = 64$ |
|---|---|---|
| $B = 2$ | $744,960$ | $3,221,225,472$ |
| $B = 3$ | $6,656$ | $1,638,400$ |

Table 2: The smallest circuit size for each bucket size and statistical security parameter, assuming $c = B$.




(a) Performance under different network.

(b) Scalability of our protocol for large inputs.

(c) Scalability of our protocol for large circuits.

Figure 7: **Efficiency of our protocol.** We test our protocol by a) changing the network bandwidth; b) increasing the input size; c) increasing the circuit size.

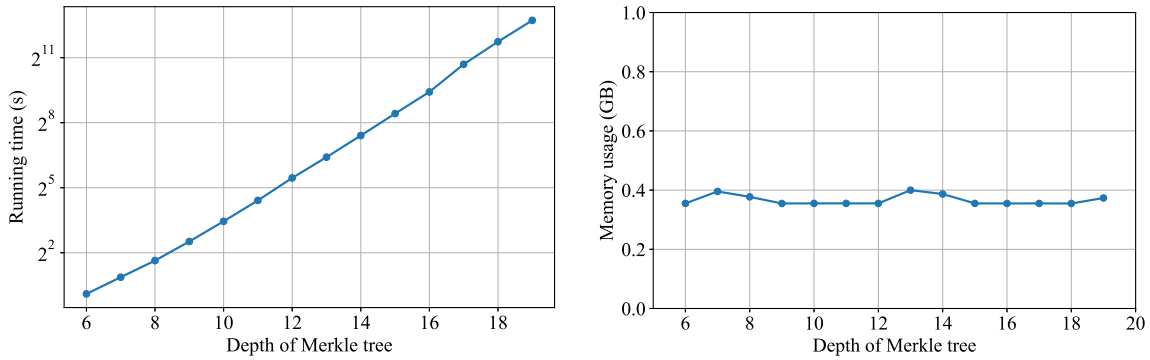
the minimal circuit size in these cases for the statistical security parameter $\rho = 40$ and $\rho = 64$. It means that we can do a batched triple check according to the Protocol 3 with parameters $B$ and $c$, when every time we finish the evaluation of at least $C$ non-linear gates.

## 5.2 Evaluation

We implement our boolean circuits protocol based on EMP-toolkit [WMK16] framework. Due to hardware support for binary-field multiplication, we use random-coefficient based batch checking protocol with and apply Fiat-Shamir heuristic to make it non-interactive. In our experiment, both the prover and the verifier are hosted on Amazon EC2 c5.2xlarge machines with memory up to 16 GiB and network bandwidth up to 5 Gbps. However, our protocol only needs a small fraction of the memory and network and thus we throttle the network and report memory usage in the evaluation. Unless specified otherwise, we parallelize our protocol using 5 threads. We used the same formulas to derive the LPN parameters that are also used in prior works [YWL+20, BCG+19a] with a target that the adversary would need $2^{128}$ steps of computation to break the assumption. All evaluation uses $\rho = 40$ and $\kappa = 128$.

**Efficiency.** To demonstrate the efficiency in both computation and communication, we show the performance of our protocol and GCZK [JKO13] in different network settings from 10 Mbps to 200 Mbps. We only compare with GCZK since their protocol is the only one that can scale to Boolean circuits with billions of gates. The Figure 7a shows the results of our experiment. Due to the low communication cost of sVOLE and our zero-knowledge protocol, our protocol can achieve a rate of 500 $ns$ per AND gate as long as the network bandwidth is larger than 40 Mbps. Even when the network is as slow as 10 Mbps, the performance is still only $1.5\mu s$ per AND gate. Compared to the GCZK [JKO13] protocol, our protocol performs far better in low-bandwidth settings because the generation of correlated OT by VOLE saves a huge amount of communication.

**Scalability.** We also measure the scalability by increasing the size of the input and the size of the circuit. The results are show in Figure 7b and Figure 7c. For both settings we do experiments




(a) The execution time of our protocol.

(b) The memory usage of our protocol.

Figure 8: Using our ZK protocol to prove knowledge of all leaves in a Merkle tree. The figures show the performance of our ZK protocol for Merkle trees of different depths. The network bandwidth is 50 Mbps.

with network throttled to 10 Mbps, 50 Mbps and 100 Mbps. From the figure, we can see that the performance of our protocol is mostly linear to the input/circuit size. This means that the efficiency preserves even for very large circuit of size more than a billion gates. In fact, due to our use of pipelining, the performance per gate does not depend on the shape of the circuit or the size of the circuit. The only potential bottleneck is the memory usage, which we will explore in detail in the next part.

**Example: Merkle trees.** We use an example to further demonstrate the efficiency and scalability of our protocol. Our example proves the knowledge of a set of Merkle-tree leaves that can be used to compute a public Merkle-tree root. In Figure 8a and Figure 8b, we show both the efficiency and the memory usage of our protocol when computing Merkle trees based on SHA-256 algorithm. We use 5 threads and a 50Mbps network in this example. The depth ranges from 6 to 19, and the number of SHA256 calls ranges from 63 to 524287. Since each SHA256 contains 22573 AND gates, our largest instance contains more than 11 billion gates. We can see that the running time is linear to the number of nodes in the tree and does not degenerate even for our largest instance. Although the circuit size to compute the root is $O(n)$ for $n$ nodes, we can evaluate the circuit in memory of size $O(\log n)$ by visiting all nodes in a DFS manner. This is also how we perform the computation. As a result, the memory usage of our protocol is very small and is below 400 MB for all cases.

## Acknowledgement

This material is based upon work supported by DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. Work of Kang Yang is supported by the National Natural Science Foundation of China (Grant No. 61932019). Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

## References

[ABF+17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority MPC

for malicious adversaries - breaking the 1 billion-gate per second barrier. In *IEEE Symposium on Security and Privacy (S&P) 2017*, pages 843–862, 2017.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 2087–2104. ACM Press, 2017.

[Ale03] Michael Alekhnovich. More on average case vs. approximation complexity. In *44th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 298–307. IEEE, 2003.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy (S&P) 2018*, pages 315–334, 2018.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Advances in Cryptology—Crypto 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, 2019.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology—Eurocrypt 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, 2016.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology—Crypto 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, 2013.

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 2105–2122. ACM Press, 2017.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM Conf. on Computer and Communications Security (CCS) 2019*, pages 291–308. ACM Press, 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *Advances in Cryptology—Crypto 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM Conf. on Computer and Communications Security (CCS) 2018*, pages 896–912. ACM Press, 2018.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology—Eurocrypt 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, 2019.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security Symposium 2014*, pages 781–796. USENIX Association, 2014.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—Crypto 1991*, LNCS, pages 420–432. Springer, 1992.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *Advances in Cryptology—Crypto 1993*, LNCS, pages 278–291. Springer, 1994.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Intl. Conference on Theory and Practice of Public Key Cryptography*, LNCS, pages 501–519. Springer, 2014.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology—Eurocrypt 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, 2015.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM Conf. on Computer and Communications Security (CCS) 2016*, pages 1292–1303. ACM Press, 2016.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology—Asiacrypt 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001.

[CDE+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod $2^k$ for dishonest majority. In *Advances in Cryptology—Crypto 2018, Part II*, volume 10992 of *LNCS*, pages 769–798. Springer, 2018.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 1825–1842. ACM Press, 2017.

[dDOS19] Cyprien de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: Using AES in picnic signatures. In *Annual International Workshop on Selected Areas in Cryptography (SAC) 2019*, LNCS, pages 669–692. Springer, 2019.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS 2013*, LNCS, pages 1–18. Springer, 2013.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.

[FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Advances in Cryptology—Eurocrypt 2017, Part II*, volume 10211 of *LNCS*, pages 225–255. Springer, 2017.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology—Eurocrypt 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, 2015.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, October 1986.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology—Eurocrypt 2013*, LNCS, pages 626–645. Springer, 2013.

[Gil99] Niv Gilboa. Two party RSA key generation. In *Advances in Cryptology—Crypto 1999*, volume 1666 of *LNCS*, pages 116–129. Springer, 1999.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 113–122. ACM Press, 2008.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In *USENIX Security Symposium 2016*, pages 1069–1083. USENIX Association, 2016.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–304. ACM Press, 1985.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 174–187. IEEE, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Advances in Cryptology—Asiacrypt 2010*, LNCS, pages 321–340. Springer, 2010.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium 2011*. USENIX Association, 2011.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In *Advances in Cryptology—Eurocrypt 2020, Part III*, volume 12107 of *LNCS*, pages 569–598. Springer, 2020.

[HOSS18] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In *Advances in Cryptology—Crypto 2018, Part III*, volume 10993 of *LNCS*, pages 3–33. Springer, 2018.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology—Crypto 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–30. ACM Press, 2007.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM Conf. on Computer and Communications Security (CCS) 2013*, pages 955–966. ACM Press, 2013.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM Conf. on Computer and Communications Security (CCS) 2018*, pages 525–537. ACM Press, 2018.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology—Crypto 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, 2015.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *ACM Conf. on Computer and Communications Security (CCS) 2016*, pages 830–842. ACM Press, 2016.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM Conf. on Computer and Communications Security (CCS) 2013*, pages 669–684. ACM Press, 2013.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In *6th Theory of Cryptography Conference—TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, 2009.

[NST17] Jesper Buus Nielsen, Thomas Schneider, and Roberto Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In *Network and Distributed System Security Symposium*. The Internet Society, 2017.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In *ACM Conf. on Computer and Communications Security (CCS) 2019*, pages 1055–1072. ACM Press, 2019.

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

[WTs+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *IEEE Symposium on Security and Privacy (S&P) 2018*, pages 926–943, 2018.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology—Crypto 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, 2019.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast Extension for Correlated OT with Small Communication, 2020.

[YWZ19] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. Cryptology ePrint Archive, Report 2019/1104, 2019. https://eprint.iacr.org/2019/1104.

[ZGK+17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A zero-knowledge version of vSQL. Cryptology ePrint Archive, Report 2017/1146, 2017. https://eprint.iacr.org/2017/1146.

[ZXZS20] J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero-knowledge proofs. In *IEEE Symposium on Security and Privacy (S&P) 2020*, 2020.

# A Other Functionalities

**Functionality $\mathcal{F}_{\mathsf{OT}}$**

On receiving $(m_0, m_1)$ with $|m_0| = |m_1|$ from $\mathsf{P_A}$ and $b \in \{0,1\}$ from $\mathsf{P_B}$, send $m_b$ to $\mathsf{P_B}$.

Figure 9: The oblivious transfer functionality between a sender $\mathsf{P_A}$ and a receiver $\mathsf{P_B}$.

We review the standard ideal functionality for oblivious transfer (OT) in Figure 9.

**Functionality $\mathcal{F}_{\mathsf{EQ}}$**

Upon receiving $V_{\mathsf{A}}$ from $\mathsf{P_A}$ and $V_{\mathsf{B}}$ from $\mathsf{P_B}$, send $(V_{\mathsf{A}} \stackrel{?}{=} V_{\mathsf{B}})$ and $V_{\mathsf{A}}$ to $\mathsf{P_B}$, and do:

- If $\mathsf{P_B}$ is honest, or is corrupted and sends continue, then send $(V_{\mathsf{A}} \stackrel{?}{=} V_{\mathsf{B}})$ to $\mathsf{P_A}$.
- If $\mathsf{P_B}$ is corrupted and sends abort, then send abort to $\mathsf{P_A}$.

Figure 10: Functionality for equality test of two elements in $\mathbb{F}_{p^r}$.

In Figure 10 we define a functionality $\mathcal{F}_{\mathsf{EQ}}$ implementing a weak equality test that reveals $\mathsf{P_A}$'s input to $\mathsf{P_B}$. This functionality is easy to be UC-realized in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model as follows: (1) $\mathsf{P_B}$ commits to $V_{\mathsf{B}}$ by calling $\mathcal{F}_{\mathsf{Com}}$; (2) $\mathsf{P_A}$ sends $V_{\mathsf{A}}$ to $\mathsf{P_B}$; (3) $\mathsf{P_B}$ outputs $(V_{\mathsf{A}} \stackrel{?}{=} V_{\mathsf{B}})$ and aborts if they are not equal, and then opens its input $V_{\mathsf{B}}$ by calling $\mathcal{F}_{\mathsf{Com}}$ again; (4) if $\mathsf{P_B}$ opened its commitment to a value $V_{\mathsf{B}}$, then $\mathsf{P_A}$ outputs $(V_{\mathsf{A}} \stackrel{?}{=} V_{\mathsf{B}})$; otherwise it aborts. The commitment functionality $\mathcal{F}_{\mathsf{Com}}$ is easy to be UC-realized in the random oracle model (see, e.g., [DKL+13]).

**Functionality $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$**

**Initialize:** Upon receiving (init) from $\mathsf{P_A}$ and $\mathsf{P_B}$, sample $\Delta \leftarrow \mathbb{F}_{p^r}$ if $\mathsf{P_B}$ is honest, and receive $\Delta \in \mathbb{F}_{p^r}$ from the adversary otherwise. Store $\Delta$, output $\Delta$ to $\mathsf{P_B}$, and ignore all subsequent (init) commands. Let $\boldsymbol{\Delta}_B \in \{0,1\}^{rm}$ be a bit-decomposition representation of $\Delta$ such that $\langle \boldsymbol{g}, \boldsymbol{\Delta}_B \rangle = \Delta$.

**Extend:** Upon receiving (extend, $u$) from $\mathsf{P_A}$ and (extend) from $\mathsf{P_B}$, this functionality operates as follows:

1. Sample $v \leftarrow \mathbb{F}_{p^r}$. If $\mathsf{P_B}$ is corrupted, instead receive $v \in \mathbb{F}_{p^r}$ from the adversary.
2. Compute $w := v + \Delta \cdot u$.
3. If $\mathsf{P_A}$ is corrupted, receive $w \in \mathbb{F}_{p^r}$ and $\boldsymbol{u} \in \mathbb{F}_p^{rm}$ from the adversary, and recompute
$$v := w - \langle \boldsymbol{g} * \boldsymbol{u}, \boldsymbol{\Delta}_B \rangle \in \mathbb{F}_{p^r},$$
where $*$ denotes the component-wise product.
4. Output $(u, w)$ to $\mathsf{P_A}$ and $v$ to $\mathsf{P_B}$.

Figure 11: Functionality for correlated oblivious product evaluation with errors (COPEe).

# B Constructing the Base sVOLE Protocol

In this section, we present a construction of the base sVOLE protocol over finite fields $\mathbb{F}_p$ and $\mathbb{F}_{p^r}$.[2] As described in Section 4, we will use this protocol to set up a small number of initial authenticated values, and then extend them to a large number of authenticated values via our sVOLE protocol with iterations.

We first describe the *correlated oblivious product evaluation with errors* (COPEe) functionality $\mathcal{F}_{\mathsf{COPEe}}$ (which extends the analogous functionality introduced by Keller et al. [KOS16] to support the subfield case we are interested in), and show how to realize this functionality in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. We then construct a protocol that UC-realizes a selective-failure version of functionality $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$ in the $\mathcal{F}_{\mathsf{COPEe}}$-hybrid model. We show that such a functionality is sufficient to guarantee the security of the ZK protocol.

## B.1 COPEe

Functionality $\mathcal{F}_{\mathsf{COPEe}}$ is described in Figure 11, where $m = \log p$. Note that the (extend) command of $\mathcal{F}_{\mathsf{COPEe}}$ can be called multiple times. In the functionality, we define a gadget vector

$$\boldsymbol{g} = \left((1, 2, \ldots, 2^{m-1}), (1, 2, \ldots, 2^{m-1}) \cdot g, \ldots, (1, 2, \ldots, 2^{m-1}) \cdot g^{r-1}\right) \in (\mathbb{F}_{p^r})^{rm},$$

and the following operation

$$\langle \boldsymbol{g}, \boldsymbol{x} \rangle = \sum_{i=0}^{r-1} \left( \sum_{j=0}^{m-1} \boldsymbol{x}[im + j] \cdot 2^j \right) \cdot g^i \in \mathbb{F}_{p^r},$$

where $\boldsymbol{x} \in \mathbb{F}^{rm}$ for a finite field $\mathbb{F} \in \{\mathbb{F}_2, \mathbb{F}_p, \mathbb{F}_{p^r}\}$. A malicious $\mathsf{P_A}$ can cause the outputs of two parties to satisfy a COPE correlation with the specific error defined in Figure 11.

We present an efficient protocol $\Pi^{p,r}_{\mathsf{COPEe}}$ shown in Figure 12 to securely compute $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. Our COPEe protocol follows the construction by Keller et al. [KOS16], which

[2] If $p = 2$ and $r = \kappa$, then our protocol is a slight improvement of the KOS OT extension protocol [KOS15].

**Protocol $\Pi^{p,r}_{\mathsf{COPEe}}$**

Let $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{F}_p$ be a pseudorandom function.

**Initialize:** This initialization procedure is executed only once.

1. For $i \in [rm]$, $\mathsf{P_A}$ samples $K^i_0, K^i_1 \leftarrow \{0,1\}^\kappa$. $\mathsf{P_B}$ samples $\Delta \leftarrow \mathbb{F}_{p^r}$ and defines its bit-composition representation as $\boldsymbol{\Delta}_B = (\Delta_1, \ldots, \Delta_{rm}) \in \{0,1\}^{rm}$.
2. For $i \in [rm]$, $\mathsf{P_A}$ sends $(K^i_0, K^i_1)$ to $\mathcal{F}_{\mathsf{OT}}$ and $\mathsf{P_B}$ sends $\Delta_i \in \{0,1\}$ to $\mathcal{F}_{\mathsf{OT}}$, which returns $K^i_{\Delta_i}$ to $\mathsf{P_B}$.

**Extend:** This procedure can be executed many times. For the $j$th input $u \in \mathbb{F}_p$ from $\mathsf{P_A}$, two parties execute the following:

3. For $i \in [rm]$, do the following in parallel:
   (a) $\mathsf{P_A}$ computes $w^i_0 := \mathsf{PRF}(K^i_0, j)$ and $w^i_1 := \mathsf{PRF}(K^i_1, j)$; and $\mathsf{P_B}$ computes $w^i_{\Delta_i} := \mathsf{PRF}(K^i_{\Delta_i}, j)$.
   (b) $\mathsf{P_A}$ sends $\tau^i := w^i_0 - w^i_1 - u \in \mathbb{F}_p$ to $\mathsf{P_B}$.
   (c) $\mathsf{P_B}$ computes $v^i := w^i_{\Delta_i} + \Delta_i \cdot \tau^i = w^i_0 - \Delta_i \cdot u \in \mathbb{F}_p$.
4. Let $\boldsymbol{v} = (v^1, \ldots, v^{rm})$ and $\boldsymbol{w} = (w^1_0, \ldots, w^{rm}_0)$ such that $\boldsymbol{w} = \boldsymbol{v} + u \cdot \boldsymbol{\Delta}_B \in \mathbb{F}^{rm}_p$.
5. $\mathsf{P_A}$ outputs $w = \langle \boldsymbol{g}, \boldsymbol{w} \rangle \in \mathbb{F}_{p^r}$ and $\mathsf{P_B}$ outputs $v = \langle \boldsymbol{g}, \boldsymbol{v} \rangle \in \mathbb{F}_{p^r}$, where $w = v + \Delta \cdot u \in \mathbb{F}_{p^r}$.

Figure 12: The COPEe protocol in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.

is in turn based on the IKNP OT extension protocol [IKNP03] and Gilboa's approach [Gil99] for oblivious product evaluation. The only difference is that the original COPEe protocol [KOS16] is further extended to support the subfield case. In the protocol $\Pi^{p,r}_{\mathsf{COPEe}}$, a malicious $\mathsf{P_A}$ may use different values w.r.t. $u$ to compute each element $\tau^i$ sent in the step 3b. Suppose that a corrupt party $\mathsf{P_A}$ uses $u^i \in \mathbb{F}_p$ to compute $\tau^i$ for $i \in [rm]$. Then, in the step 4 of protocol $\Pi^{p,r}_{\mathsf{COPEe}}$, we will instead have that $\boldsymbol{w} = \boldsymbol{v} + \boldsymbol{u} * \boldsymbol{\Delta}_B \in \mathbb{F}^{rm}_p$, where $*$ means element-wise multiplication. where $\boldsymbol{u} = (u^1, \ldots, u^{rm})$. This results in

$$w = v + \langle \boldsymbol{g} * \boldsymbol{u}, \boldsymbol{\Delta}_B \rangle \in \mathbb{F}_{p^r}.$$

The malicious behavior is not prevented, and instead is modeled in functionality $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$. In the proof of Appendix B.2, we show that such functionality is sufficient to guarantee the security of our sVOLE protocol. Following the proof [KOS16], one can easily obtain the following lemma.

**Lemma 3.** *Let* $\mathsf{PRF}$ *be a pseudorandom function. Protocol* $\Pi^{p,r}_{\mathsf{COPEe}}$ *shown in Figure 12 UC-realizes functionality* $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$ *in the* $\mathcal{F}_{\mathsf{OT}}$*-hybrid model.*

Note that the OT messages $\{(K^i_0, K^i_1)\}$ are uniform in the COPEe protocol, and thus we can replace $\mathcal{F}_{\mathsf{OT}}$ with a random oblivious transfer (ROT) functionality $\mathcal{F}_{\mathsf{ROT}}$. This allows us to obtain better efficiency.

## B.2 Base sVOLE

We first define an sVOLE functionality with the selective failure leakage of global key $\Delta$ (denoted by $\mathcal{F}^{p,r}_{\mathsf{LsVOLE}}$). Functionality $\mathcal{F}^{p,r}_{\mathsf{LsVOLE}}$ is the same as $\mathcal{F}_{\mathsf{sVOLE}}$ described in Figure 2, except that if $\mathsf{P_A}$ is corrupted, then the adversary is additionally allowed to make the following selective failure query *only once for each* extend *execution* before the output is sent:

**Protocol $\Pi^{p,r}_{\mathsf{base\text{-}sVOLE}}$**

1. $\mathsf{P_A}$ and $\mathsf{P_B}$ send $\mathsf{init}$ to functionality $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$, which returns $\Delta \in \mathbb{F}_{p^r}$ to $\mathsf{P_B}$.
2. $\mathsf{P_A}$ samples $u_i \leftarrow \mathbb{F}_p$ for $i \in [n]$ and $a_h \leftarrow \mathbb{F}_p$ for $h \in [r]$. For $i \in [n]$, $\mathsf{P_A}$ sends $(\mathsf{extend}, u_i)$ to $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$, and $\mathsf{P_B}$ sends $(\mathsf{extend})$ to $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$, which returns $w_i \in \mathbb{F}_{p^r}$ to $\mathsf{P_A}$ and $v_i \in \mathbb{F}_{p^r}$ to $\mathsf{P_B}$ such that $w_i = v_i + \Delta \cdot u_i$. For $h \in [r]$, both parties also call $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$ on respective input $(\mathsf{extend}, a_h)$ and $(\mathsf{extend})$, where $\mathsf{P_A}$ gets $c_h \in \mathbb{F}_{p^r}$ and $\mathsf{P_B}$ obtains $b_h \in \mathbb{F}_{p^r}$ such that $c_h = b_h + \Delta \cdot a_h$.
3. $\mathsf{P_B}$ samples $\chi_1, \ldots, \chi_n \leftarrow \mathbb{F}_{p^r}$, and sends them to $\mathsf{P_A}$. Then $\mathsf{P_A}$ computes $x := \sum_{i=1}^{n} \chi_i \cdot u_i + \sum_{h=1}^{r} a_h \cdot g^{h-1} \in \mathbb{F}_{p^r}$ and $z := \sum_{i=1}^{n} \chi_i \cdot w_i + \sum_{h=1}^{r} c_h \cdot g^{h-1} \in \mathbb{F}_{p^r}$, and then sends $(x, z)$ to $\mathsf{P_B}$.
4. $\mathsf{P_B}$ computes $y := \sum_{i=1}^{n} \chi_i \cdot v_i + \sum_{h=1}^{r} b_h \cdot g^{h-1} \in \mathbb{F}_{p^r}$, and then check that $z = y + x \cdot \Delta$. If the check fails, $\mathsf{P_B}$ aborts.
5. $\mathsf{P_A}$ outputs $\boldsymbol{u} := (u_1, \ldots, u_n)$ and $\boldsymbol{w} := (w_1, \ldots, w_n)$; $\mathsf{P_B}$ outputs $\boldsymbol{v} = (v_1 \ldots, v_n)$.

Figure 13: Base sVOLE protocol in the $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$-hybrid model.

- Wait for the adversary to input $(\mathsf{guess}, S)$ where $S$ efficiently describes a subset of $\mathbb{F}_{p^r}$. If $\Delta \in S$, then send $\mathsf{success}$ to the adversary and continue. Otherwise, send $\mathsf{abort}$ to both parties and abort.

Compared to the functionality $\mathcal{F}_{\mathsf{sVOLE}}$ introduced before, the selective failure queries on $\Delta$ may leak a small amount of bits of $\Delta$, where an incorrect guess will be caught. This leakage can be eliminated without increasing much cost by using the technique by Nielse et al. [NST17]. On the other hand, even if we do not eliminate this leakage, it will not impact the overall security of the ZK protocol, following the observation in prior works [KOS15, CDE+18, YWZ19]. In particular, the probability that all the selective failure queries are successful is bounded by $|S|/p^r$ (if there are more than one query then $S$ is taken as the intersection of all sets). Conditioned on this event, the min-entropy of global key $\Delta$ is reduced to $\log |S|$. Therefore, the overall probability of the adversary that made successful selective failure queries on $\Delta$ and passed the batched check on the opening of authenticated values is bounded by

$$\frac{|S|}{p^r} \cdot \frac{1}{2^{\log |S|}} = \frac{1}{p^r},$$

which is the same as that without such leakage.

In Figure 13, we show a protocol $\Pi^{p,r}_{\mathsf{base\text{-}sVOLE}}$ that UC-realizes $\mathcal{F}^{p,r}_{\mathsf{LsVOLE}}$ in the $\mathcal{F}_{\mathsf{COPEe}}$-hybrid model. The construction of our protocol is inspired by the OT extension protocol [KOS15] and the SPDZ-like protocol MASCOT [KOS16], and particularly supports the subfield case. In the protocol $\Pi^{p,r}_{\mathsf{base\text{-}sVOLE}}$, $\mathsf{P_B}$ performs a *correlation check* (i.e., steps 3 and 4) to check that the outputs of two parties satisfy a correct correlation (i.e., $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$). From the construction, we note that the base sVOLE protocol allows $\mathsf{P_A}$ to choose its output vector $\boldsymbol{u}$. Similarly, $\mathsf{P_B}$ can send a uniform $\mathsf{seed} \in \{0,1\}^\kappa$ to $\mathsf{P_A}$, and then both parties can simply derive the coefficients $\{\chi_i\}$ from $\mathsf{seed}$ using a random oracle.

Below, we prove the security of our base sVOLE protocol, based on the proof idea of the MASCOT authentication protocol [KOS16].

**Theorem 4.** *Protocol $\Pi^{p,r}_{\mathsf{base\text{-}sVOLE}}$ UC-realizes $\mathcal{F}^{p,r}_{\mathsf{sVOLE}}$ in the $\mathcal{F}^{p,r}_{\mathsf{COPEe}}$-hybrid model. In particular, no PPT environment $\mathcal{Z}$ can distinguish the real world execution from the ideal world execution, except with probability at most $(r \log p)^2/p^r$.*

*Proof.* We first consider the case of a malicious $\mathsf{P_A}$ and then consider the case of a malicious $\mathsf{P_B}$. In each case, we construct a PPT simulator $\mathcal{S}$ which runs a PPT adversary $\mathcal{A}$ as a subroutine, and emulates the functionality $\mathcal{F}_{\mathsf{COPEe}}^{p,r}$. We always implicitly assume that $\mathcal{S}$ passes all communication between $\mathcal{A}$ and $\mathcal{Z}$.

**Malicious $\mathsf{P_A}$.** $\mathcal{S}$ has access to functionality $\mathcal{F}_{\mathsf{LsVOLE}}^{p,r}$, and interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{COPEe}}^{p,r}$, and receives the values $(w_i, \boldsymbol{u}_i) \in \mathbb{F}_{p^r} \times \mathbb{F}_p^{rm}$ for $i \in [n]$ and $(c_h, \boldsymbol{a}_h) \in \mathbb{F}_{p^r} \times \mathbb{F}_p^{rm}$ for $h \in [r]$ from adversary $\mathcal{A}$.
   In the (semi)-honest case, we have that $\boldsymbol{u}_i = (u_i, \ldots, u_i)$ for some $u_i \in \mathbb{F}_p$ and $\boldsymbol{a}_h = (a_h, \ldots, a_h)$ for some $a_h \in \mathbb{F}_p$.

2. $\mathcal{S}$ samples $\chi_1, \ldots, \chi_n \leftarrow \mathbb{F}_{p^r}$ and sends them to adversary $\mathcal{A}$. Then, $\mathcal{S}$ receives $x \in \mathbb{F}_{p^r}$ and $z \in \mathbb{F}_{p^r}$ from $\mathcal{A}$. Simulator $\mathcal{S}$ computes an adversarially chosen error
$$e_z := z - \sum_{i=1}^{n} \chi_i \cdot w_i - \sum_{h=1}^{r} c_h \cdot g^{h-1} \in \mathbb{F}_{p^r}.$$

3. $\mathcal{S}$ computes a guess set $S_\Delta$ as follows:
   - Solve the following equation:
$$\left\langle \boldsymbol{g} \cdot x - \boldsymbol{g} * \sum_{i=1}^{n} \chi_i \cdot \boldsymbol{u}_i - \boldsymbol{g} * \sum_{h=1}^{r} \boldsymbol{a}_h \cdot g^{h-1}, \boldsymbol{\Delta}_B \right\rangle = e_z. \tag{1}$$
   - For each solution $\boldsymbol{\Delta}_B$, compute $\Delta := \langle \boldsymbol{g}, \boldsymbol{\Delta}_B \rangle$ and add $\Delta$ into the set $S_\Delta$.

4. $\mathcal{S}$ sends $(\mathsf{guess}, S_\Delta)$ to functionality $\mathcal{F}_{\mathsf{LsVOLE}}^{p,r}$. If receiving $\mathsf{abort}$ from $\mathcal{F}_{\mathsf{LsVOLE}}^{p,r}$, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ continues the simulation.

5. $\mathcal{S}$ computes another set $\tilde{S}_{\tilde{\Delta}}$ as follows:
   - Solve the following equation:
$$\left\langle \boldsymbol{g} \cdot x - \boldsymbol{g} * \sum_{i=1}^{n} \chi_i \cdot \boldsymbol{u}_i - \boldsymbol{g} * \sum_{h=1}^{r} \boldsymbol{a}_h \cdot g^{h-1}, \tilde{\boldsymbol{\Delta}}_B \right\rangle = 0. \tag{2}$$
   - For each solution $\tilde{\boldsymbol{\Delta}}_B$, compute $\tilde{\Delta} := \langle \boldsymbol{g}, \tilde{\boldsymbol{\Delta}}_B \rangle$ and add $\tilde{\Delta}$ into the set $\tilde{S}_{\tilde{\Delta}}$.

6. If $\tilde{S}_{\tilde{\Delta}}$ only involves a single entry 0, then $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ chooses any non-zero element $\tilde{\Delta} \in \tilde{S}_{\tilde{\Delta}}$ and for $i \in [n]$, computes
$$u_i := \tilde{\Delta}^{-1} \cdot \langle \boldsymbol{g} * \boldsymbol{u}_i, \tilde{\boldsymbol{\Delta}}_B \rangle, \tag{3}$$
   where $\tilde{\Delta} = \langle \boldsymbol{g}, \tilde{\boldsymbol{\Delta}}_B \rangle$. We will show that $u_i$ is unique over all possible $\tilde{\Delta}$ in set $\tilde{S}_{\tilde{\Delta}}$.

7. For $i \in [n]$, $\mathcal{S}$ computes an adversarially chosen error $\boldsymbol{e}_i = \boldsymbol{u}_i - (u_i, \ldots, u_i) \in \mathbb{F}_p^{rm}$, and then computes $w_i' := w_i - \langle \boldsymbol{g} * \boldsymbol{e}_i, \boldsymbol{\Delta}_B \rangle \in \mathbb{F}_{p^r}$ for any $\boldsymbol{\Delta}_B$ such that $\Delta = \langle \boldsymbol{g}, \boldsymbol{\Delta}_B \rangle \in S_\Delta$. Then, $\mathcal{S}$ sends $\boldsymbol{u} = (u_1, \ldots, u_n)$ and $\boldsymbol{w} = (w_1', \ldots, w_n')$ to functionality $\mathcal{F}_{\mathsf{LsVOLE}}^{p,r}$.

The simulation for the protocol transcript is straightforward, and thus we focus on other analysis (particularly the probability of aborting and uniqueness of $u_i$). In the following, we first consider the case of $p = 2$, and later will discuss the case that $p > 2$ is a prime.

In the real protocol execution, the correlation check has the following equation:

$$x \cdot \Delta = z - y = \sum_{i=1}^{n} \chi_i \cdot (w_i - v_i) + \sum_{h=1}^{r} (c_h - b_h) \cdot g^{h-1} + e_z. \tag{4}$$

In the case of malicious $\mathsf{P_A}$, we have that $w_i - v_i = \langle \boldsymbol{g} * \boldsymbol{u}_i, \boldsymbol{\Delta}_B \rangle$ for $i \in [n]$ and $c_h - b_h = \langle \boldsymbol{g} * \boldsymbol{a}_h, \boldsymbol{\Delta}_B \rangle$ for $h \in [r]$. Thus, we can rewrite the equation (4) as follows:

$$\begin{aligned} & x \cdot \Delta - \sum_{i=1}^{n} \chi_i \cdot \langle \boldsymbol{g} * \boldsymbol{u}_i, \boldsymbol{\Delta}_B \rangle - \sum_{h=1}^{r} \langle \boldsymbol{g} * \boldsymbol{a}_h, \boldsymbol{\Delta}_B \rangle \cdot g^{h-1} = e_z \\ \Leftrightarrow & \left\langle \boldsymbol{g} \cdot x - \boldsymbol{g} * \sum_{i=1}^{n} \chi_i \cdot \boldsymbol{u}_i - \boldsymbol{g} * \sum_{h=1}^{r} \boldsymbol{a}_h \cdot g^{h-1}, \boldsymbol{\Delta}_B \right\rangle = e_z. \end{aligned}$$

The solution set $S_\Delta$ corresponds to the adversary's guess on global key $\Delta$. In the following, we will prove that this set is unique. Under the condition, we obtain that the probability of aborting in the ideal world execution is the same as that in the real world execution.

For any two different solutions $\Delta, \Delta' \in S_\Delta$, we define $\tilde{\Delta} = \Delta - \Delta' \in \mathbb{F}_{p^r}$ and thus $\tilde{\boldsymbol{\Delta}}_B = \boldsymbol{\Delta}_B - \boldsymbol{\Delta}'_B \in \{0,1\}^{rm}$. From the equation (1), we easily obtain that the equation (2) holds. This also shows that the set $S_\Delta$ for equation (1) is an affine subspace of $\mathbb{F}_{p^r}$. Note that the solution set $\tilde{S}_{\tilde{\Delta}}$ about equation (2) is a linear space parallel to $S_\Delta$. If there is only one solution for the equation (1), then $\tilde{S}_{\tilde{\Delta}}$ includes only one zero entry. In this case, simulator $\mathcal{S}$ aborts, and the probability, that the real protocol execution does not abort, is at most $1/p^r$.

Let $\boldsymbol{a}_h = a_h + \boldsymbol{e}'_h \in \mathbb{F}_p^{rm}$ for $h \in [r]$ and some $a_h \in \mathbb{F}_p$ such that

$$\sum_{h=1}^{r} \langle \boldsymbol{g} \cdot a_h - \boldsymbol{g} * \boldsymbol{a}_h, \tilde{\boldsymbol{\Delta}}_B \rangle \cdot g^{h-1} = 0, \tag{5}$$

where $\boldsymbol{e}'_h$ is an adversarially chosen error and $\tilde{\Delta} = \langle \boldsymbol{g}, \tilde{\boldsymbol{\Delta}}_B \rangle \in \tilde{S}_{\tilde{\Delta}}$ is used to compute $u_i$ for $i \in [n]$ in the equation (3). Clearly, the equations (3) and (5) provide a solution for $x = \sum_{i=1}^{n} \chi_i \cdot u_i + \sum_{h=1}^{r} a_h \cdot g^{h-1} \in \mathbb{F}_{p^r}$ such that $\sum_{h=1}^{r} \langle \boldsymbol{g} \cdot a_h - \boldsymbol{g} * \boldsymbol{a}_h, \tilde{\boldsymbol{\Delta}}_B \rangle \cdot g^{h-1} = 0$ and

$$\langle \boldsymbol{g} \cdot u_i - \boldsymbol{g} * \boldsymbol{u}_i, \tilde{\boldsymbol{\Delta}}_B \rangle = 0 \text{ for all } i \in [n]$$

hold for some $\tilde{\Delta} = \langle \boldsymbol{g}, \tilde{\boldsymbol{\Delta}}_B \rangle \in \tilde{S}_{\tilde{\Delta}}$. Below, we need to prove that $\{u_i\}_{i \in [n]}$ computed by the equation (3) is the unique solution for a sufficiently large subspace of $\tilde{S}_{\tilde{\Delta}}$. Now, we assume that for some $l \in \mathbb{N}$, for each $f \in [l]$, there exists a different set $\{u_{f,i}\}_{i \in [n]}$ along with the set $\{a_{f,h}\}_{h \in [r]}$ such that $x = \sum_{i=1}^{n} \chi_i \cdot u_{f,i} + \sum_{h=1}^{r} a_{f,h} \cdot g^{h-1}$ and

$$\langle \boldsymbol{g} \cdot u_{f,i} - \boldsymbol{g} * \boldsymbol{u}_i, \tilde{\boldsymbol{\Delta}}^f_B \rangle = 0 \text{ for all } i \in [n] \text{ and } \sum_{h=1}^{r} \langle \boldsymbol{g} \cdot a_{f,h} - \boldsymbol{g} * \boldsymbol{a}_h, \tilde{\boldsymbol{\Delta}}^f_B \rangle \cdot g^{h-1} = 0, \tag{6}$$

for all $\tilde{\Delta}^f = \langle \boldsymbol{g}, \tilde{\boldsymbol{\Delta}}^f_B \rangle \in \tilde{S}_f \subseteq \tilde{S}_{\tilde{\Delta}}$ such that $|\tilde{S}_f| > 1$. The condition of $|\tilde{S}_f| > 1$ is required for adversary $\mathcal{A}$ to pass the correlation check successfully with probability more than $1/p^r$. Since $\tilde{S}_f$ clearly is a linear space for all $f \in [l]$ and $\tilde{S}_f \cap \tilde{S}_{f'} = \{0\}$ from the definition, and $|\tilde{S}_{\tilde{\Delta}}| \leq p^r$ by definition, we have that $l \leq r \log p$.

Let $f \neq f' \in [l]$. From the equation (2) and $x = \sum_{i=1}^{n} \chi_i \cdot u_{f',i} + \sum_{h=1}^{r} a_{f',h} \cdot g^{h-1}$, we have that the following holds:

$$\sum_{i=1}^{n} \chi_i \cdot u_{f',i} \cdot \tilde{\Delta}^f - \sum_{i=1}^{n} \chi_i \cdot \langle \boldsymbol{g} * \boldsymbol{u}_i, \tilde{\boldsymbol{\Delta}}_B^f \rangle + \sum_{h=1}^{r} a_{f',h} \cdot \tilde{\Delta}^f \cdot g^{h-1} - \sum_{h=1}^{r} \langle \boldsymbol{g} * \boldsymbol{a}_h, \tilde{\boldsymbol{\Delta}}_B^f \rangle \cdot g^{h-1} = 0.$$

Using the equation (6), we obtain that

$$\sum_{i=1}^{n} \chi_i \cdot (u_{f',i} - u_{f,i}) \cdot \tilde{\Delta}^f + \sum_{h=1}^{r} (a_{f',h} - a_{f,h}) \cdot \tilde{\Delta}^f \cdot g^{h-1} = 0. \tag{7}$$

By definition, there exists some $j \in [n]$ such that $u_{f,j} \neq u_{f',j}$. Furthermore, there are at least two values for $\tilde{\Delta}^f \in \tilde{S}_f$, and thus we assume that in the above equation $\tilde{\Delta}^f \neq 0$. Thus, $(u_{f',j} - u_{f,j}) \cdot \tilde{\Delta}^f \neq 0$. Note that $\chi_1, \ldots, \chi_n$ are sampled uniformly at random and independent from the other values involved in the equation (7). Therefore, the equation (7) holds with probability at most $1/p^r$. There are less than $l^2 \leq (r \log p)^2$ pairs $f \neq f' \in [l]$. The overall probability is bounded by $(r \log p)^2 / p^r$.

We have established that there exists a unique solution $u_i$ for $i \in [n]$. This means that for all $\tilde{\Delta} = \langle \boldsymbol{g}, \tilde{\boldsymbol{\Delta}}_B \rangle \in \tilde{S}_{\tilde{\Delta}}$, we have that $\langle \boldsymbol{g} \cdot u_i - \boldsymbol{g} * \boldsymbol{u}_i, \tilde{\boldsymbol{\Delta}}_B \rangle = 0$ for $i \in [n]$. Therefore, we obtain that $\langle \boldsymbol{g} * \boldsymbol{e}_i, \tilde{\boldsymbol{\Delta}}_B \rangle = 0$ for all $i \in [n]$. If there exists two different $\Delta, \Delta' \in S_\Delta$ such that $\langle \boldsymbol{g} * \boldsymbol{e}_i, \boldsymbol{\Delta}_B \rangle \neq \langle \boldsymbol{g} * \boldsymbol{e}_i, \boldsymbol{\Delta}'_B \rangle$ for some $i \in [n]$ where $\langle \boldsymbol{g}, \boldsymbol{\Delta}_B \rangle = \Delta$ and $\langle \boldsymbol{g}, \boldsymbol{\Delta}'_B \rangle = \Delta'$, then we define $\tilde{\boldsymbol{\Delta}}_B := \boldsymbol{\Delta}_B - \boldsymbol{\Delta}'_B$ such that $\tilde{\Delta} = \langle \boldsymbol{g}, \tilde{\boldsymbol{\Delta}}_B \rangle \in \tilde{S}_{\tilde{\Delta}}$ and $\langle \boldsymbol{g} * \boldsymbol{e}_i, \tilde{\boldsymbol{\Delta}}_B \rangle \neq 0$. This is contradict with $\langle \boldsymbol{g} * \boldsymbol{e}_i, \tilde{\boldsymbol{\Delta}}_B \rangle = 0$. This concludes that $\langle \boldsymbol{g} * \boldsymbol{e}_i, \boldsymbol{\Delta}_B \rangle$ is a unique value for all possible $\Delta = \langle \boldsymbol{g}, \boldsymbol{\Delta}_B \rangle \in S_\Delta$, and can be computed by the simulator (and the adversary) using any $\Delta \in S_\Delta$. In the real protocol execution, adversary $\mathcal{A}$ can compute $w'_i := w_i - \langle \boldsymbol{g} * \boldsymbol{e}_i, \boldsymbol{\Delta}_B \rangle$ for $i \in [n]$ just as that computed by simulator $\mathcal{S}$. Together with that $w_i = v_i + \langle \boldsymbol{g} * \boldsymbol{u}_i, \boldsymbol{\Delta}_B \rangle$, we have that

$$w'_i = v_i + \langle \boldsymbol{g} * \boldsymbol{u}_i, \boldsymbol{\Delta}_B \rangle - \langle \boldsymbol{g} * \boldsymbol{e}_i, \boldsymbol{\Delta}_B \rangle = v_i + \Delta \cdot u_i.$$

We now discuss the case of a prime $p > 2$. The main difference to the case of $p = 2$ is that the canonical maps between $\Delta \in \mathbb{F}_{p^r}$ and $\boldsymbol{\Delta}_B \in \{0,1\}^{rm}$ are not bijective. This implies that the solutions of equations (1) and (2) are not necessarily vectors of bits rather than elements of $\mathbb{F}_p$. Following the proof [KOS16, Lemma 2], we have that if $\tilde{S}_f$ includes at least two vectors that only consist of bits, which is necessary for the adversary to pass the correlation check with probability more than $1/p^r$, then it has dimension at least 1 for all $f \in [l]$. We also have the fact that $\tilde{S}_{\tilde{\Delta}}$ has dimension at most $r \log p$ and $\tilde{S}_f \cap \tilde{S}_{f'} = \{0\}$ for $f \neq f' \in [1, l]$ by definition. Together, we obtain that $l \leq r \log p$ as above.

**Malicious $\mathsf{P_B}$.** $\mathcal{S}$ has access to functionality $\mathcal{F}_{\mathsf{LsVOLE}}^{p,r}$, and interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{COPEe}}^{p,r}$, and receives the values $\Delta$, $v_i$ for $i \in [n]$ and $b_h$ for $h \in [r]$ from $\mathcal{A}$.
2. After receiving coefficients $\chi_1, \ldots, \chi_n \in \mathbb{F}_{p^r}$, $\mathcal{S}$ samples $x \leftarrow \mathbb{F}_{p^r}$, computes $y := \sum_{i=1}^{n} \chi_i \cdot v_i + \sum_{h=1}^{r} b_h \cdot g^{h-1} \in \mathbb{F}_{p^r}$, and computes $z := y + x \cdot \Delta \in \mathbb{F}_{p^r}$. Then $\mathcal{S}$ sends $(x, z)$ to adversary $\mathcal{A}$.
3. $\mathcal{S}$ defines $\boldsymbol{v} = (v_1, \ldots, v_n)$ and sends $\boldsymbol{v} \in \mathbb{F}_{p^r}^n$ to functionality $\mathcal{F}_{\mathsf{LsVOLE}}^{p,r}$.

In the real protocol execution, the elements $a_h$ for all $h \in [r]$ output by $\mathcal{F}_{\mathsf{COPEe}}^{p,r}$ are uniform in $\mathbb{F}_p$. Therefore, $\sum_{h=1}^{r} a_h \cdot g^{h-1}$ is uniform in $\mathbb{F}_{p^r}$, and thus $x = \sum_{i=1}^{n} \chi_i \cdot u_i + \sum_{h=1}^{r} a_h \cdot g^{h-1}$ is uniformly random in $\mathbb{F}_{p^r}$. We obtain that the simulation is perfect. It is easy to see that the outputs of two parties have the same distribution between the real world execution and the ideal world execution. □