Moz$\mathbb{Z}_{2^k}$arella: Efficient Vector-OLE and Zero-Knowledge Proofs Over $\mathbb{Z}_{2^k}$*

Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl

Aarhus University
{cbaum,braun,almun,peter.scholl}@cs.au.dk

Abstract. Zero-knowledge proof systems are usually designed to support computations for circuits over $\mathbb{F}_2$ or $\mathbb{F}_p$ for large p, but not for computations over $\mathbb{Z}_{2^k}$, which all modern CPUs operate on. Although $\mathbb{Z}_{2^k}$-arithmetic can be emulated using prime moduli, this comes with an unavoidable overhead. Recently, Baum et al. (CCS 2021) suggested a candidate construction for a designated-verifier zero-knowledge proof system that natively runs over $\mathbb{Z}_{2^k}$. Unfortunately, their construction requires preprocessed random vector oblivious linear evaluation (VOLE) to be instantiated over $\mathbb{Z}_{2^k}$. Currently, it is not known how to efficiently generate such random VOLE in large quantities.

In this work, we present a maliciously secure, VOLE extension protocol that can turn a short seed-VOLE over $\mathbb{Z}_{2^k}$ into a much longer, pseudorandom VOLE over the same ring. Our construction borrows ideas from recent protocols over finite fields, which we non-trivially adapt to work over $\mathbb{Z}_{2^k}$. Moreover, we show that the approach taken by the QuickSilver zero-knowledge proof system (Yang et al. CCS 2021) can be generalized to support computations over $\mathbb{Z}_{2^k}$. This new VOLE-based proof system, which we call QuarkSilver, yields better efficiency than the previous zero-knowledge protocols suggested by Baum et al. Furthermore, we implement both our VOLE extension and our zero-knowledge proof system, and show that they can generate 13–50 million VOLEs per second for 64 bit to 256 bit rings, and evaluate 1.3 million 64 bit multiplications per second in zero-knowledge.

1 Introduction

Zero-knowledge (ZK) proofs allow a prover to convince a verifier that some statement is true, without revealing any additional information. They are a fundamental tool in cryptography with a wide range of applications. A common way of expressing statements used in ZK is with *circuit satisfiability*, where the prover and verifier hold some circuit $\mathcal{C}$, and the prover proves that she knows a witness w such that $\mathcal{C}(w) = 1$. Typically, $\mathcal{C}$ is an arithmetic circuit defined over

* This article is the full version of an article with the same name [Bau+22], © IACR 2022, which appeared at Crypto 2022. The version published by Springer-Verlag is available at https://doi.org/10.1007/978-3-031-15985-5_12.

a finite field such as $\mathbb{F}_2$ or $\mathbb{F}_p$ for a large prime p, but the same idea works for any finite ring.

A recent line of work [Wen+21; Yan+21; Bau+21b; DIO21] builds highly scalable zero-knowledge proofs based on vector oblivious linear evalution, or VOLE. VOLE is a two-party protocol often used in secure computation settings, which allows a receiver holding Δ to learn a secret linear function $\mathbf{w} - \Delta \cdot \mathbf{u} = \mathbf{v}$ of a sender's private inputs $\mathbf{u}, \mathbf{w}$. VOLE-based ZK protocols have the key feature that the overhead of the prover is very small: compared with the cost of evaluating the circuit $\mathcal{C}$ in the clear, few additional computational or memory resources are needed. This allows proofs to scale to handle very large statements, such as proving properties of complex programs. On the other hand, potential drawbacks of using VOLE are that the communication complexity is typically linear in the size of $\mathcal{C}$ – unlike SNARKs (e.g. [Mal+19; Ben+19]) and MPC-in-the-head techniques (e.g. [Ame+17]), which can be sublinear – and proofs are only verifiable by a single, designated verifier.

VOLE Constructions. In a length-n VOLE protocol over some ring R, the sender has input two vectors $\mathbf{u}, \mathbf{w} \in R^n$, while the receiver has input $\Delta \in R$, and receives as output $\mathbf{v} \in R^n$ as defined above. In applications such as ZK proofs, it is actually enough to construct *random VOLEs*, or VOLE correlations, where both parties' inputs are chosen at random. The most efficient approaches for generating random VOLE are based on the method of Boyle et al. [Boy+18], which relies on an arithmetic variant of the learning parity with noise (LPN) assumption. The protocol has the key feature that the communication cost is *sublinear* in the output length, n.

The original protocol of [Boy+18] has only semi-honest security (or malicious security using expensive, generic 2-PC techniques). Later, dedicated maliciously secure protocols over fields were developed [Boy+19; Wen+21], which essentially match the cost of the underlying semi-honest protocols, by using lightweight consistency checks for verifying honest behavior. In general, these protocols assume that R is a finite field.

ZK Based on VOLE. The state-of-the-art, VOLE-based protocol for proving circuit satisfiability in zero-knowledge is the QuickSilver protocol. QuickSilver, which builds upon the previous Line-Point ZK [DIO21] protocol, works for circuits over any finite field $\mathbb{F}_q$, and has a communication cost of essentially 1 field element per multiplication gate. Concretely, QuickSilver achieves a throughput of up to 15.8 million AND gates per second for a Boolean circuit, or 8.9 million multiplication gates for an arithmetic circuit over the 61-bit Mersenne prime field. Another approach is the Mac'n'Cheese protocol [Bau+21b], which can also achieve an amortized cost as small as 1 field element, but with slightly worse computational costs and round complexity.

ZK Over Rings. While most ZK protocols are based on circuits over fields, it can in certain applications be more desirable to work with circuits over a finite ring such as $\mathbb{Z}_{2^k}$. For instance, to prove a property of an existing program (such

as proving a program contains a bug, or does not violate some safety property) the program logic and computations must all be emulated using a circuit. Since CPUs perform arithmetic in $\mathbb{Z}_{2^k}$, this is a natural choice of ring that leads to a simpler translation of program code into a satisfiable circuit $\mathcal{C}$.

Unfortunately, not many existing ZK proof systems can natively support computations over rings. The recent work of [Bau+21a] gave the first ZK protocol over $\mathbb{Z}_{2^k}$ based on VOLE over $\mathbb{Z}_{2^k}$, obtaining a proof system with a communication cost of $O(1)$ ring elements per multiplication gate (for large rings), asymptotically matching QuickSilver over large fields. However, a major drawback of their protocols is that they require maliciously secure VOLE over $\mathbb{Z}_{2^k}$, which is much more expensive to build: the only known instantiation of this [Sch18] would increase the concrete communication of their ZK protocol by 1–2 orders of magnitude. Finally, another approach to zero-knowledge proof systems over rings has been proposed based on SNARKs [GNS21]. When using $\mathbb{Z}_{2^k}$, this work obtains a designated-verifier SNARK, however, the scheme has not been implemented, and suffers from a dependency on expensive, public-key cryptography, as in many field-based SNARKs.

1.1 Contributions

In this work, we address the question of building efficient protocols for VOLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. Firstly, we show how to build a maliciously secure VOLE protocol over $\mathbb{Z}_{2^k}$, with efficiency comparable to state-of-the-art protocols over finite fields [Boy+19; Wen+21]. Our protocol introduces new consistency checks for verifying correctness of VOLE extension, which are tailored to overcome the difficulties of working with the ring $\mathbb{Z}_{2^k}$. Secondly, using our VOLE over $\mathbb{Z}_{2^k}$, we show how to adapt the QuickSilver protocol [Yan+21] to the ring setting, obtaining an efficient ZK protocol called QuarkSilver that is dedicated to proving circuit satisfiability over $\mathbb{Z}_{2^k}$. Here, we extend techniques from the MPC world [Cra+18] to be suitable for our ZK proof. Finally, we implemented and benchmarked both our VOLE and ZK protocols to demonstrate their performance. In a high-bandwidth, low-latency setting, our implementation achieves a throughput of 13–50 million VOLEs per second for 64 bit to 256 bit rings with 40 bit statistical security while transmitting only ≈ 1 bit per VOLE. Our QuarkSilver implementation is able to compute and verify 1.3 million 64 bit multiplications per second.

1.2 Our Techniques

Below, we expand on our contributions, the techniques involved and some more relevant background.

Challenge of Working in $\mathbb{Z}_{2^k}$. Before delving into our protocols, we first briefly recap the main challenges when working with rings like $\mathbb{Z}_{2^k}$, compared with finite fields. When using VOLE for zero-knowledge, VOLE is used to *commit*

the prover to its inputs and intermediate wire values in the circuit. This is possible by viewing each VOLE output $M[x] = \Delta \cdot x + K[x]$ as an information-theoretic homomorphic MAC in the input x.

When working over a finite field, it's easy to see that if a malicious prover can come up with a valid MAC $M[\overline{x}]$ on an input $\overline{x} \neq x$, for the same key $K[x]$, then the prover can recover the MAC key Δ from the relation:

$$M[x] - M[\overline{x}] = \Delta \cdot (x - \overline{x})$$

However, this relies on $x - \overline{x}$ being invertible, which is usually not the case when working over a ring such as $\mathbb{Z}_{2^k}$. Indeed, if $x - \overline{x} = 2^{k-1}$, then the prover can forge a MAC $M[\overline{x}]$ with probability $1/2$, since $M[x] - M[\overline{x}] \bmod 2^k$ now only depends on the least significant bit of Δ.

The SPD$\mathbb{Z}_{2^k}$ protocol [Cra+18] for multi-party computation showed how to work around this issue by extending the modulus to 2^{k+s}, for some statistical security parameter s. This way, it can be shown that the lower s bits of the key Δ are still enough to protect the integrity of the lower k bits of the message x.

Indeed, this was exactly the type of MAC scheme used in the recent work on conversions and ZK over rings [Bau+21a]. However, as in the SPD$\mathbb{Z}_{2^k}$ protocols, further challenges arise when handling more complex protocols for verifying computation on MACed values.

Maliciously Secure VOLE Extension in $\mathbb{Z}_{2^k}$. Current state-of-the-art VOLE protocols all stem from the approach of Boyle et al. [Boy+18], which builds a *pseudorandom correlation generator* based on (variants of) the *learning parity with noise* (LPN) assumption. This approach exploits the fact that sparse LPN errors can be used to compress secret-sharings of pseudorandom vectors, allowing the two parties to generate a long, pseudorandom instance of a VOLE correlation in a succinct manner.

These protocols proceed by first constructing a protocol for *single-point* VOLE, where the sender's input vector has only a single non-zero entry. Then, the single-point VOLE protocol is repeated t times, to obtain a t-point VOLE where the sender's input is viewed as a long, sparse, LPN error vector. Finally, by combining t-point VOLE and the LPN assumption, the parties can locally transform this into pseudorandom VOLE by appling a linear mapping.

Using this blueprint leads to (random) VOLE protocols with communication much smaller than the output length. This can be seen as a form of *VOLE extension*, where in the first step, a small "seed" VOLE of length $m \ll n$ is used to create the single-point VOLEs, and then extended into a longer VOLE of length n. In the Wolverine protocol [Wen+21], it was additionally observed that when repeating this process, it can greatly help communication if m of the n extended outputs are reserved and used to bootstrap the next iteration of the protocol, saving generation of fresh seed VOLEs.

With semi-honest security, the above approach can easily be instantiated over rings, following the protocols of [Sch+19; Boy+19]. When adapting this protocol to malicious security, our main technical challenge is that previous works over

fields [Boy+19; Wen+21] used a consistency check to verify correctness of the outputs, which involved taking random linear combinations over the field. Due to the existence of zero divisors, this technique does not directly translate to $\mathbb{Z}_{2^k}$. One possible approach, similarly to the MAC scheme described above, is to increase the size of the ring to, say, $\mathbb{Z}_{2^{k+s}}$, and use computations in the larger ring to ensure that the VOLEs are correct modulo 2^k. However, the problem is, it would then no longer be compatible with the bootstrapping technique of [Wen+21]: to check consistency, the seed VOLE must be in the larger ring $\mathbb{Z}_{2^{k+s}}$, however, since the outputs are only in $\mathbb{Z}_{2^k}$, they can't then be used as a seed for the next execution! One solution would be to start with an even larger ring ($\mathbb{Z}_{2^{k+2s}}$), and keep decreasing the ring size after each iteration, but this would be far too expensive when done repeatedly.

Instead, we take a different approach. First, we adopt a hash-based check from [Boy+19], which verifies correctness of a puncturable pseudorandom function based on a GGM tree, created during the protocol. This hash check (which we optimize by using universal hashing instead of a cryptographic hash function) works over rings as well as fields, however, it does not suffice to ensure consistency of the entire protocol. On top of this, we incorporate a linear combination check, however, one with binary coefficients instead of coefficients in the large ring. This type of check can be used over a ring, but allows a cheating prover to try to bypass the check and cheat successfully with probability 1/2. Nevertheless, we show that by allowing some additional leakage in the single-point VOLE functionality, we can still simulate the protocol with this check. For our final VOLE protocol, this leakage implies that a few noise coordinates of the LPN error vector may have leaked.

While previous protocols also allowed a limited form of leakage [Boy+19; Wen+21], in this case, ours is more serious since entire noise coordinates can be leaked with probability 1/2. To counter this, we analyze the state-of-the-art attacks on LPN, and show how to adjust the parameters and increase the noise rate accordingly.

Similarly to [Wen+21], we focus on using the "primal" form of LPN, which was also used for semi-honest VOLE over $\mathbb{Z}_{2^k}$ in [Sch+19]. While the "dual" form of LPN, as considered in [Boy+18; Boy+19; CRR21], achieves lower communication costs (and does not rely on bootstrapping), it involves a more costly matrix multiplication, which is expensive to implement. In [Boy+19], dual-LPN was instantiated using quasi-cyclic codes to achieve $\tilde{O}(n)$ complexity, but this approach does not readily adapt to rings instead of fields; it is plausible that the fast, LDPC-based dual-LPN variant proposed in [CRR21] can be adapted to work over rings, but the security of this assumption has not been analyzed thoroughly.

Efficient Zero-Knowledge via QuarkSilver in $\mathbb{Z}_{2^k}$. Given VOLE, the standard approach to obtaining a ZK proof is using the homomorphic MAC scheme described above. There, the prover first commits to the input $\mathbf{w}$ as well as all intermediate circuit wire values of $\mathcal{C}(\mathbf{w})$. Then, the prover must show consis-

tency of all the wire values and that the output wire indeed contains 1. Since the MACs are linearly homomorphic, the main challenge is verifying multiplications. In QuickSilver [Yan+21], to verify that committed values x, y, z satisfy $x \cdot y = z$, the parties locally compute a quadratic function on their MACs and MAC keys, obtaining a new value which has a consistent MAC only if the multiplication is correct.

The catch is that this new MAC relation being checked leads to a quadratic equation in the secret key Δ, instead of linear as before, which is chosen by a possibly dishonest prover. If this quadratic equation has a root in Δ, then the check passes. In the field case, this is not a problem as there are no more than two solutions to a quadratic equation, so we obtain a soundness error of $2/|\mathbb{F}|$. However, with rings, there can be many solutions. For instance, with

$$f(X) = aX^2 + bX + c \pmod{2^k},$$

if $a = 2^{k/2}$ and $b = c = 0$ then any multiple of $2^{k/4}$ is a possible choice for X, i.e. the check would erroneously pass for $2^{3k/4}$ choices of Δ. To remedy this, we reduce the number of valid solutions by working modulo 2^ℓ for some $\ell > k$, and adding the constraint on the solution that $\Delta \in \{0, \ldots, 2^s - 1\}$, where s is a statistical security parameter.

An additional challenge is that when checking a batch of multiplications, we actually check a random linear combination of a large number of these equations, which again leads to complications with zero divisors. By carefully analyzing the number of bounded solutions to equations of this type, and extending techniques from SPD$\mathbb{Z}_{2^k}$ [Cra+18] for handling linear combinations over rings, we show that it suffices to choose $\ell \approx k + 2(\sigma + \log \sigma)$ to achieve $2^{-\sigma}$ failure probability in the check. Overall, we obtain a communication complexity of ℓ bits per input and multiplication gate in the circuit.

2 Preliminaries

2.1 Notation

We use lower case, bold symbols for vectors $\mathbf{x}$ and upper case, bold symbols for matrices $\mathbf{A}$. We use κ as the computational and σ as the statistical security parameter. In our UC functionalities and proofs, $\mathcal{Z}$ denotes the environment, and $\mathcal{S}$ is the simulator, while $\mathcal{A}$ will refer to the adversary.

2.2 Vector OLE

Vector OLE (VOLE) is a two party functionality between a sender $\mathsf{P_S}$ and a receiver $\mathsf{P_R}$ to obtain correlated random vectors of the following form: $\mathsf{P_S}$ obtains two vectors $\mathbf{u}, \mathbf{w}$, and $\mathsf{P_R}$ gets a random scalar Δ and a random vector $\mathbf{v}$ so that $\mathbf{w} = \Delta \cdot \mathbf{u} + \mathbf{v}$ holds.

We parameterize the functionality with two values ℓ and s such that $s \leq \ell$. The scalar Δ is sampled from $\mathbb{Z}_{2^s}$, and the vectors $\mathbf{u}, \mathbf{v}, \mathbf{w}$ are sampled from

$\mathbb{Z}_{2^\ell}^n$ where n denotes the size of the correlation. We require that the equation $\mathbf{w} = \Delta \cdot \mathbf{u} + \mathbf{v}$ holds modulo 2^ℓ. The ideal functionality is described in Figure 1.

As in SPDZ$_{2^k}$ [Cra+18], we can implement $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ using the oblivious transfer protocol (OT) of [Sch18]. Basing VOLE on OT has the drawback of quadratic communication costs in the ring size, since it requires one OT of size ℓ bit for each of the ℓ bits of a ring element. Hence, we would use this approach only once to create a set of base VOLEs. Then we can use the more efficient protocol presented in Section 4 to repeatedly generate large batches to VOLEs.

VOLE for $\mathbb{Z}_{2^k}$: $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$

Let $\ell \geq s$.

Init This method is the first to be called by the parties. On input (Init) from both parties proceed as follows:

1. If $\mathsf{P_R}$ is honest, sample $\Delta \in_R \mathbb{Z}_{2^s}$ and send Δ to $\mathsf{P_R}$.
2. If $\mathsf{P_R}$ is corrupt, receive $\Delta \in \mathbb{Z}_{2^s}$ from $\mathcal{S}$.
3. Δ is stored by the functionality.

All further (Init) queries are ignored.

Extend On input (Extend, n) from both parties proceed as follows:

1. If $\mathsf{P_R}$ is honest, sample $\mathbf{v} \in_R \mathbb{Z}_{2^\ell}^n$. Otherwise receive $\mathbf{v} \in_R \mathbb{Z}_{2^\ell}^n$ from $\mathcal{S}$.
2. If $\mathsf{P_S}$ is honest, sample $\mathbf{u} \in_R \mathbb{Z}_{2^\ell}^n$ and compute $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v} \in \mathbb{Z}_{2^\ell}^n$. Otherwise receive $\mathbf{u} \in \mathbb{Z}_{2^\ell}^n$ and $\mathbf{w} \in \mathbb{Z}_{2^\ell}^n$ from $\mathcal{S}$ and then recompute $\mathbf{v} := \mathbf{w} - \Delta \cdot \mathbf{u} \in \mathbb{Z}_{2^\ell}^n$
3. Send $(\mathbf{u}, \mathbf{w})$ to $\mathsf{P_S}$ and $\mathbf{v}$ to $\mathsf{P_R}$.

Global-key Query If $\mathsf{P_S}$ is corrupted, receive (Guess, Δ') from $\mathcal{S}$ with $\Delta' \in \mathbb{Z}_{2^s}$. If $\Delta' = \Delta$, send success to $\mathsf{P_S}$ and ignore subsequent global-key queries. Otherwise, send abort to both parties and abort.

Fig. 1. Ideal functionality VOLE over $\mathbb{Z}_{2^k}$.

2.3 Equality Test

In our work, we use an equality test functionality $\mathcal{F}_{\mathsf{EQ}}$ (Figure 2) between two parties $\mathcal{P}, \mathcal{V}$ where $\mathcal{V}$ learns the input of $\mathcal{P}$. The equality check functionality can be implemented using a simple commit-and-open protocol, see e.g. [Wen+21]. When using a hash function with 2κ bit output (modeled as random oracle) to implement the commitment scheme, the equality check of ℓ bit values can be implemented with $\ell + 3\kappa$ bit of communication.

Equality Test: $\mathcal{F}_{\mathsf{EQ}}$

On input $V_{\mathcal{P}}$ from $\mathcal{P}$ and $V_{\mathcal{V}}$ from $\mathcal{V}$:

1. Send $V_{\mathcal{P}}$ and $(V_{\mathcal{P}} \stackrel{?}{=} V_{\mathcal{V}})$ to $\mathcal{V}$.
2. If $\mathcal{V}$ is honest and $V_{\mathcal{P}} = V_{\mathcal{V}}$, or $\mathcal{V}$ is corrupted and sends $\mathsf{continue}$, then send $(V_{\mathcal{P}} \stackrel{?}{=} V_{\mathcal{V}})$ to $\mathcal{P}$
3. If $\mathcal{V}$ is honest and $V_{\mathcal{P}} \neq V_{\mathcal{V}}$, or $\mathcal{V}$ is corrupted and sends abort, then send abort to $\mathcal{P}$.

Fig. 2. Ideal functionality for equality tests.

2.4 Zero-Knowledge Proofs of Knowledge

In Figure 3 we provide an ideal functionality for zero-knowledge proofs. The functionality implies the standard definition of a ZKPoK as it is:

Complete because whenever $\mathcal{C}(\mathbf{w}) = 1$ then an honest verifier will accept.
Knowledge Sound because it outputs true iff $\mathcal{P}$ inputs, and thus knows, a satisfying assignment $\mathbf{w}$ for $\mathcal{C}$.
Zero-Knowledge because nothing beyond the check $\mathcal{C}(\mathbf{w}) \stackrel{?}{=} 1$ is leaked to $\mathcal{V}$.

Zero-Knowledge Functionality $\mathcal{F}_{\mathsf{ZK}}^{k}$

Prove: On input (prove, $\mathcal{C}$, $\mathbf{w}$) from $\mathcal{P}$ and (verify, $\mathcal{C}$) from $\mathcal{V}$ where $\mathcal{C}$ is a circuit over $\mathbb{Z}_{2^k}$ and $\mathbf{w} \in \mathbb{Z}_{2^k}^n$ for some $n \in \mathbb{N}$: Send true to $\mathcal{V}$ iff $\mathcal{C}(\mathbf{w}) = 1$, and false otherwise.

Fig. 3. Ideal functionality for zero-knowledge proofs for circuit satisfiability.

2.5 The LPN Assumption Over Rings

The *Learning Parity with Noise* (LPN) assumption [Blu+94] states that, given the noisy dot product of many public vectors $\mathbf{a}_i$ with a secret vector $\mathbf{s}$, the result is indistinguishable from a vector of random values. Adding noise to indices is done by adding a noise vector $\mathbf{e}$ at the end, consisting of random values.

We rely on the following arithmetic variant of LPN over a ring $\mathbb{Z}_M$, as also considered in [Boy+18; Sch+19].

Definition 1 (LPN). *Let $\mathcal{D}_{n,t}^{M}$ be a distribution over $\mathbb{Z}_M^n$ such that for any $t, n, M \in \mathbb{N}$, $\mathrm{Im}(\mathcal{D}_{n,t}^{M}) \in \mathbb{Z}_M^n$. Let G be a probabilistic code generation algorithm*

such that $\mathsf{G}(m, n, M)$ *outputs a matrix* $\mathbf{A} \in \mathbb{Z}_M^{m \times n}$. *Let parameters* m, n, t *be implicit functions of security parameter* κ. *The* $\mathsf{LPN}^{\mathsf{G}}_{m,n,t,M}$ *assumptions states that:*

$$\{(\mathbf{A}, \mathbf{x}) \mid \mathbf{A} \leftarrow \mathsf{G}(m, n, M), \mathbf{s} \in_R \mathbb{Z}_M^m, \mathbf{e} \leftarrow \mathcal{D}_{n,t}^M, \mathbf{x} := \mathbf{s} \cdot \mathbf{A} + \mathbf{e}\}$$
$$\approx_C \{(\mathbf{A}, \mathbf{x}) \mid \mathbf{A} \leftarrow \mathsf{G}(m, n, M), \mathbf{x} \in_R \mathbb{Z}_M^n\}.$$

There exists two flavours of the LPN assumption; the primal (Definition 1) and the dual (Definition 15, Appendix B).

Informally, the main advantage of the *primal* version of LPN is that there exist practical (and implemented) constructions of the LPN-friendly codes required for this. Specifically, one can choose the code matrix $\mathbf{A}$ from a family of codes G supporting *linear-time* matrix-vector multiplication, such as *d-local linear codes* so that each column of $\mathbf{A}$ has exactly d non-zero entries. According to [Ale03], the hardness of LPN for local linear codes is well-established. Its main disadvantage however, is that its output size can be at most quadratic in the size of the seed, as intuitively, a higher stretch would make it significantly easier for an adversarial verifier to guess enough noiseless coordinates to allow efficient decoding via Gaussian Elimination [AG11].

The main advantage of the *dual* variant is that it allows for an arbitrary polynomial stretch. However, the compressive mapping used within the dual variant cannot have constant locality and is more challenging to instantiate. Recently, Silver [CRR21] proposed an instantiation of dual-LPN based on structured LDPC codes, which have been practically implemented over finite fields, and may plausibly also work over rings.

Dealing with Reduction Attacks Over Rings. When working over a ring $\mathbb{Z}_M$ instead of a finite field, we must take care that the presence of zero divisors does not weaken security. For instance, a simple reduction attack was pointed out in [Liu+22], where noise values can become zero after reducing modulo a factor of M (for instance, in $\mathbb{Z}_{2^k}$, reducing the LPN sample modulo 2 cuts the number of noisy coordinates in half, significantly reducing security). To mitigate this attack, we always sample non-zero entries of the error vector $\mathbf{e}$ and matrix $\mathbf{A}$ to be in $\mathbb{Z}_M^*$, that is, invertible mod M.[1] While [Liu+22] did not consider the effect on the matrix $\mathbf{A}$, we observe that if $\mathbf{A}$ is sparse then its important to ensure that its sparsity cannot also be decreased through reduction.[2] With these countermeasures, we are not aware of any attacks on LPN in $\mathbb{Z}_M$ that perform better than the field case.

We elaborate below on our choice of primal-LPN distribution.

[1] This countermeasure was missing from the original version of this paper, before [Liu+22] was available.

[2] On the other hand, the LPN secret $\mathbf{s}$ must *not* be chosen over $\mathbb{Z}_M^*$, but instead uniformly over $\mathbb{Z}_M$, since if e.g. $\mathbf{s}$ was known to be odd over $\mathbb{Z}_{2^k}$ then solving the reduced instance modulo 2 would be trivial.

Choice of Matrix over $\mathbb{Z}_M$. We choose a random, sparse matrix $\mathbf{A}$ with d non-zero entries per column. We choose each non-zero entry randomly from $\mathbb{Z}_M^*$, to ensure that it remains non-zero after reduction modulo any factor of M. We fix the sparsity to $d = 10$, as in previous works [Boy+18; Sch+19; Wen+21], which according to [App+17; Zic17] suffices to ensure that $\mathbf{A}$ has a large dual distance, which implies the LPN samples are unbiased [CRR21].

Noise Distribution in $\mathbb{Z}_M$. The noise distribution $\mathcal{D}_{n,t}^M$ is chosen to have t expected non-zero coordinates. This can be done on expectation with a Bernoulli distribution, where each coordinate is either zero, or non-zero (and uniform otherwise) with probability t/n. In our applications, we instead use an exact noise weight, where $\mathcal{D}_{n,t}^M$ fixes t non-zero coordinates in the length-n vector.

Invertible Noise Terms. When working over a ring $\mathbb{Z}_M$, we sample the non-zero noise values to be in $\mathbb{Z}_M^*$, that is, invertible mod M. This prevents the reduction attack mentioned above, which would otherwise reduce the expected noise weight by a factor of two for $M = 2^k$.

Uniform vs Regular Noise Patterns. For fixed-weight noise, we speak of *two types* of error; *regular* or *uniform*. We call uniform errors the case where $\mathcal{D}_{n,t}^M$ is the uniform distribution over all weight-t vectors of $\mathbb{Z}_M^n$ with non-zero values in $\mathbb{Z}_M^*$. Implementing LPN-based PCGs with uniform errors has previously been investigated by [Yan+20; Sch+19]. It is commonly implemented by utilising a sub-protocol to place a single non-zero value within a vector of length $n' \ll n$ and then using Cuckoo hashing to generate a uniform distribution over n from several of these smaller vectors, ending up with the t points distributed randomly across the n coordinates.

Our construction uses a *regular* noise distribution for the primal-LPN instance. Here, the noise vector in $\mathbb{Z}_M^n$ is divided into t blocks of length $\lfloor n/t \rfloor$, such that each block has exactly one non-zero coordinate. Generally, using LPN with regular errors is practically more efficient than for uniform errors [Yan+20; Wen+21].

3 Single-Point Vector OLE

Single-point VOLE is a specialized functionality that generates a VOLE correlation $\mathbf{w} = \Delta \cdot \mathbf{u} + \mathbf{v}$ (see Section 2.2) where $\mathbf{u}$ has only one non-zero coordinate $\alpha \in [n]$. We consider a variant where u_α is not only non-zero, but additionally also required to be invertible.

We present an ideal functionality for single-point VOLE $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$ in Figure 4. In the functionality, $\mathsf{P_S}$ obtains $\mathbf{u}, \mathbf{w} \in \mathbb{Z}_{2^\ell}^n \times \mathbb{Z}_{2^\ell}^n$, and $\mathsf{P_R}$ gets $\Delta, \mathbf{v} \in \mathbb{Z}_{2^s} \times \mathbb{Z}_{2^\ell}^n$. As in the full VOLE functionality $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ we allow $\mathsf{P_S}$ to attempt to guess Δ. Additionally, $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$ also allows $\mathsf{P_R}$ to obtain leakage on the non-zero index:

Single-Point VOLE for $\mathbb{Z}_{2^\ell}$: $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$

This functionality extends the functionality $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ (Figure 1). In addition to the methods (Init) and (Extend), it also provides the method (SP-Extend) and a modified global-key query.

SP-Extend On input (SP-Extend, n) with $n \in \mathbb{N}$ from both parties the functionality proceeds as follows:

1. Sample $\mathbf{u} \in_R \mathbb{Z}^n_{2^\ell}$ with a single entry invertible modulo 2^ℓ and zeros everywhere else, $\mathbf{v} \in_R \mathbb{Z}^n_{2^\ell}$, and compute $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v} \in \mathbb{Z}^n_{2^\ell}$.
2. If $\mathsf{P_S}$ is corrupted, receive $\mathbf{u} \in \mathbb{Z}^n_{2^\ell}$ with at most one non-zero entry and $\mathbf{w} \in \mathbb{Z}^n_{2^\ell}$ from $\mathcal{S}$, and recompute $\mathbf{v} := \mathbf{w} - \Delta \cdot \mathbf{u}$.
3. If $\mathsf{P_R}$ is corrupted:
   (a) Receive a set $I \subseteq [n]$ from $\mathcal{S}$. Let $\alpha \in [n]$ be the index of the non-zero entry $\mathbf{u}$, and let $\beta := u_\alpha$. If $I = \{\alpha\}$, then send (success, β) to $\mathsf{P_R}$. If $\alpha \in I$ and $|I| > 1$, then send success to $\mathsf{P_R}$ and continue. Otherwise send abort to both parties and abort.
   (b) Receive either (continue) or (query, J) from $\mathcal{S}$. If (continue) was received, continue with Step 3c. If (query, J) with $J \subset [n]$ and $|J| = \frac{n}{2}$ was received and $\alpha \in J$, then send α to $\mathcal{S}$. Otherwise, send abort to all parties, and abort.
   (c) Receive $\mathbf{v} \in \mathbb{Z}^n_{2^\ell}$ from $\mathcal{S}$, and recompute $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v}$.
4. Send $(\mathbf{u}, \mathbf{w})$ to $\mathsf{P_S}$ and $\mathbf{v}$ to $\mathsf{P_R}$.

Global-key Query If $\mathsf{P_S}$ is corrupted, receive (Guess, Δ', s') from $\mathcal{S}$ with $s' \leq s$ and $\Delta' \in \mathbb{Z}_{2^{s'}}$. If $\Delta' = \Delta \pmod{2^{s'}}$, send success to $\mathsf{P_S}$. Otherwise, send abort to both parties and abort.

Fig. 4. Ideal functionality for a leaky single-point VOLE.

1. $\mathsf{P_R}$ is allowed to guess a set $I \subseteq [n]$ that should contain the index α. Upon correct guess, if $|I| = 1$ then it learns $\mathbf{u}_\alpha$ while if $|I| > 1$ the functionality continues. If $\alpha \notin I$ then the functionality aborts.
2. $\mathsf{P_R}$ is also allowed a second query for a set $J \subset [n]$ that might contain α where $|J| = n/2$. If $\mathsf{P_R}$ guesses correctly then the functionality outputs α, while it aborts otherwise.

The leakage is somewhat inherent to our protocol which we use to realize $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$.

Protocol Overview. Our protocol $\Pi^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ (Figure 5) achieves active security using consistency checks inspired by the constructions from [Boy+19] and [Wen+21]. We now give a high-level overview.

As a setup, we assume functionalities $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{EQ}}$. For $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ we assume that $\mathsf{P_R}$ called (Init) already, thus setting Δ. Additionally, we require two

pseudorandom generators (PRGs; with certain extra properties that we clarify in Section 3.1) to create a GGM tree. Recall, the GGM construction [GGM84] builds a PRF from a length-doubling PRG, by recursively expanding a PRG seed into 2 seeds, defining a complete binary tree where each of the n leaves is one evaluation of the PRF. We use this to build a puncturable PRF, where a subset of intermediate tree nodes is given out, enabling evaluating the PRF at all-but-one of the points in the domain.

The sender $\mathsf{P_S}$ begins by picking a random index α from $[n]$, and β randomly from $\mathbb{Z}_{2^\ell}^*$. This defines the vector $\mathbf{u}$ where $\mathbf{u}_\alpha = \beta$ and every other index is 0. $\mathsf{P_S}$ and $\mathsf{P_R}$ use a single VOLE from $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ to authenticate β, resulting in the receiver holding γ and the sender holding δ, β such that $\delta = \Delta \cdot \beta + \gamma$.

To extend this correlation to the whole vector $\mathbf{u}$, $\mathsf{P_R}$ computes a GGM tree with $2n$ leaves. We consider all n leaves that are "left children" of their parent as comprising the vector $\mathbf{v}$. Using $\log_2(n)$ instances of $\mathcal{F}_{\mathsf{OT}}$, $\mathsf{P_S}$ learns all "right children" as well as all of the "left children" except the one at position α – meaning that the sender learns $\mathbf{v}$ for all indices except α. $\mathsf{P_S}$ now sets $\mathbf{w}_i = \mathbf{v}_i$ for $i \neq \alpha$. This gives a valid correlation on these $n-1$ positions, because since $\mathbf{u}_i = 0$ for $i \neq \alpha$, we have that $\mathbf{w}_i = \Delta \cdot \mathbf{u}_i + \mathbf{v}_i$.

What remains in the protocol is for $\mathsf{P_S}$ to learn $\mathbf{w}_\alpha = \Delta \cdot \mathbf{u}_\alpha + \mathbf{v}_\alpha$ without revealing α and β to $\mathsf{P_R}$. Using the output of the VOLE instance, if $\mathsf{P_R}$ computes $d \leftarrow \gamma - \sum_{j=1}^{n} \mathbf{v}_j$ and sends d to $\mathsf{P_S}$, then $\mathsf{P_S}$ can compute

$$\begin{aligned}
\mathbf{w}_\alpha &= \delta - d - \sum_{j \in [n] \setminus \{\alpha\}} \mathbf{w}_j \\
&= \delta - \Big(\gamma - \sum_{i \in [n]} \mathbf{v}_i\Big) - \sum_{j \in [n] \setminus \{\alpha\}} \mathbf{w}_j \\
&= \delta - \gamma + \mathbf{v}_\alpha = \Delta \cdot \beta + \mathbf{v}_\alpha
\end{aligned}$$

which is exactly the missing value for the correlation. While this protocol can somewhat easily be proven secure against a dishonest $\mathsf{P_S}$ (assuming that the hybrid functionalities are actively secure), a corrupted $\mathsf{P_R}$ can cheat in two ways:

1. It can provide inconsistent GGM tree values to the $\mathcal{F}_{\mathsf{OT}}$ instances, thus leading to unpredictable protocol behavior.
2. It can construct d incorrectly.

To ensure a "somewhat consistent" GGM tree (and inputs to $\mathcal{F}_{\mathsf{OT}}$) we use a check that sacrifices all the leaves that are "right children". Here, $\mathsf{P_R}$ has to send a random linear combination of these, over a binary extension field, with $\mathsf{P_S}$ choosing the coefficients. The check makes sure that if it passes, then the "left children" are consistent for every choice of α that would have made $\mathsf{P_S}$ not abort. This reduces arbitrary leakage to an essentially unavoidable selective failure attack (due to the use of $\mathcal{F}_{\mathsf{OT}}$).

To prevent the second attack, the sender and receiver use an additional VOLE from $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ and perform a random linear combination check to ensure correctness of the value d. Due to the binary coefficients used in the linear combination

over $\mathbb{Z}_{2^\ell}$, our check only has soundness 1/2. This, however, suffices to prove security if we relax the functionality by allowing a corrupt receiver to learn α with probability 1/2. This way, in the simulation in our security proof, if the challenge vector $\boldsymbol{\chi}$ is such that the receiver passes the check despite cheating, the simulator can still extract a valid input using its knowledge of α.

The full protocol is presented in Figure 5. Before proving security of it, we first recap the Puncturable PRF from GGM construction and its security properties.

3.1 Checking Consistency of the GGM Construction

We use the GGM [GGM86] construction to implement a puncturable PRF F with domain $[n]$ and range $\{0,1\}^\kappa$.

In a puncturable PRF (PPRF), one party P_1 generates a PRF key k, and then both parties engage in a protocol where the second party P_2 obtains a punctured key $k\{\alpha\}$ for an index $\alpha \in [n]$ of its choice. With $k\{\alpha\}$, it is possible for P_2 to evaluate F at all points $[n] \setminus \{\alpha\}$ so that $F(k,i) = F(k\{\alpha\}, i)$ for $i \neq \alpha$, while nothing about $F(k,\alpha)$ is revealed. More formally:

Definition 2 (Adapted from [Boy+19]). *A puncturable pseudorandom function (PPRF) with keyspace $\mathcal{K}$, domain $[n]$ and range $\{0,1\}^\kappa$ is a pseudorandom function F with an additional keyspace $\mathcal{K}_p$ and 3 PPT algorithms* KeyGen, Gen, PuncEval *such that*

KeyGen *on input 1^κ outputs a random key $k \in \mathcal{K}$.*
Gen *on input n, k outputs $\{F(k,i), k\{i\}\}_{i\in[n]}$ where $k\{i\} \in \mathcal{K}_p$.*
PuncEval *on input $n, \alpha, k\{\alpha\}$ outputs $\mathbf{v}^\alpha$ such that $\mathbf{v}^\alpha \in (\{0,1\}^\kappa)^n$.*

where $F(k,i) = \mathbf{v}^\alpha_i$ for all $i \neq \alpha$ and no PPT adversary $\mathcal{A}$, given $n, \alpha, k\{\alpha\}$ as input, can distinguish $F(k,\alpha)$ from a uniformly random value in $\{0,1\}^\kappa$ except with probability $\mathsf{negl}(\kappa)$.

For simplicity, we describe the algorithms for domains of size $n = 2^h$ for some $h \in \mathbb{N}$. By pruning the tree appropriately, the procedures can be adapted to support domain sizes that are not powers of two. Throughout the coming sections, we let $\alpha_1, \ldots, \alpha_h$ be the bit decomposition of $\alpha = \sum_{i=0}^{h-1} 2^i \cdot \alpha_{h-i}$, and let $\overline{\alpha}_i$ denote the complement. Let κ be a computational and σ be a statistical security parameter. Define $\sigma' := \sigma + 2\log n$ and let $\mathsf{G}\colon \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ and $\mathsf{G}'\colon \{0,1\}^\kappa \to \mathbb{Z}_{2^\ell} \times \mathbb{F}_{2^{\sigma'}}$ be two PRGs.

Recall that to achieve malicious security when generating a PPRF key in our protocol, we use the redundancy introduced from extending the domain to size $2n$, and check consistency by letting the receiver provide a hash of all the right leaves of the GGM tree. In order for the right leaves of the GGM tree to fix a unique tree, we require the PRG used for the final layer $\mathsf{G}'\colon \{0,1\}^\kappa \to \mathbb{Z}_{2^\ell} \times \mathbb{F}_{2^{\sigma'}}$ to satisfy the *right-half injectivity* property[3] as defined below.

[3] As noted in [Boy+19], this can be replaced with a weaker notion of right-half collision resistance, which is easier to achieve in practice.

Single-Point VOLE for $\mathbb{Z}_{2^\ell}$: $\Pi^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$

For the (Init) and (Extend) operations, the parties simply query $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$.

SP-Extend For (SP-Extend, n): Let $h := \lceil \log n \rceil$ and $\sigma' := \sigma + 2h$.

1. The parties send (Extend, 1) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$. $\mathsf{P_S}$ receives $a, c \in \mathbb{Z}_{2^\ell}$ and $\mathsf{P_R}$ receives $b \in \mathbb{Z}_{2^\ell}$ such that $c = \Delta \cdot a + b \pmod{2^\ell}$ holds.
2. $\mathsf{P_S}$ samples $\alpha \in_R [n], \beta \in_R \mathbb{Z}^*_{2^\ell}$ and lets $\mathbf{u} \in \mathbb{Z}^n_{2^\ell}$ be the vector with $u_\alpha = \beta$ and $u_i = 0$ for all $i \neq \alpha$.
3. $\mathsf{P_S}$ sets $\delta := c$ and sends $a' := \beta - a \in \mathbb{Z}_{2^\ell}$ to $\mathsf{P_R}$. $\mathsf{P_R}$ computes $\gamma := b - \Delta \cdot a' \in \mathbb{Z}_{2^\ell}$. Now, $\delta = \Delta \cdot \beta + \gamma \pmod{2^\ell}$.
4. $\mathsf{P_R}$ computes $k \leftarrow \mathsf{GGM.KeyGen}(1^\kappa)$, runs $(\mathbf{v}, \mathbf{t}, (\overline{K}^i_0, \overline{K}^i_1)_{i\in[h]}, \overline{K}^{h+1}_1) \leftarrow \mathsf{GGM.Gen}(n, k)$, and sends $\overline{K}^{h+1} := \overline{K}^{h+1}_1 \in \mathbb{F}_{2^{\sigma'}}$ to $\mathsf{P_S}$.
5. Write $\alpha = \sum_{i=0}^{h-1} 2^i \cdot \alpha_{h-i}$, for $\alpha_i \in \{0,1\}$. For $i \in [h]$, the parties call $\mathcal{F}_{\mathsf{OT}}$ where $\mathsf{P_S}$, acting as the receiver, inputs $\overline{\alpha}_i$ and $\mathsf{P_R}$ inputs $(\overline{K}^i_0, \overline{K}^i_1)_{i\in[h]}$ to $\mathcal{F}_{\mathsf{OT}}$. $\mathsf{P_S}$ receives $\overline{K}^i := \overline{K}^i_{\overline{\alpha}_i}$.
6. *Check the GGM tree:*
   (a) $\mathsf{P_S}$ samples $\boldsymbol{\xi} \in_R \mathbb{F}^n_{2^{\sigma'}}$ and sends $\boldsymbol{\xi}$ to $\mathsf{P_R}$.[a]
   (b) $\mathsf{P_R}$ computes $\Gamma := \langle \boldsymbol{\xi}, \mathbf{t} \rangle \in \mathbb{F}_{2^{\sigma'}}$ and sends Γ to $\mathsf{P_S}$.
   (c) $\mathsf{P_S}$ runs $\mathbf{v}^\alpha \leftarrow \mathsf{GGM.PuncEval}(n, \alpha, (\overline{K}^i)_{i\in[h+1]})$ followed by $\mathsf{GGM.Check}(n, \alpha, (\overline{K}^i)_{i\in[h+1]}, \boldsymbol{\xi}, \Gamma)$. If the latter returns $\perp$, $\mathsf{P_S}$ aborts. Otherwise it has obtained $(v_j)_{j\in[n]\setminus\{\alpha\}}$.
7. $\mathsf{P_R}$ sends $d := \gamma - \sum_{j=1}^n v_j \in \mathbb{Z}_{2^\ell}$ to $\mathsf{P_S}$. $\mathsf{P_S}$ defines $\mathbf{w} \in \mathbb{Z}^n_{2^\ell}$ such that $w_j := v_j$ for $j \in [n]\setminus\{\alpha\}$ and $w_\alpha := \delta - d - \sum_{\substack{1\le j\le n \\ j\neq\alpha}} w_j$. Then $\mathbf{w} = \Delta \cdot \mathbf{u} + \mathbf{v}$.
8. *Check consistency of d:*
   (a) The parties send (Extend, 1) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$. $\mathsf{P_S}$ receives $x, z \in \mathbb{Z}_{2^\ell}$ and $\mathsf{P_R}$ receives $y^* \in \mathbb{Z}_{2^\ell}$ such that $z = \Delta \cdot x + y^* \pmod{2^\ell}$ holds.
   (b) $\mathsf{P_S}$ samples $\boldsymbol{\chi} \in_R \{0,1\}^n$ with $\mathrm{HW}(\boldsymbol{\chi}) = \frac{n}{2}$ and sends it to $\mathsf{P_R}$.[b]
   (c) $\mathsf{P_S}$ computes $x^* := \chi_\alpha \cdot \beta - x \in \mathbb{Z}_{2^\ell}$ and sends x^* to $\mathsf{P_R}$. $\mathsf{P_R}$ computes $y := y^* - \Delta \cdot x^* \in \mathbb{Z}_{2^\ell}$. Then $z = y + \Delta \cdot \chi_\alpha \cdot \beta$.
   (d) $\mathsf{P_S}$ computes $V_{\mathsf{P_S}} := \sum_{i=1}^n \chi_i \cdot w_i - z$, and $\mathsf{P_R}$ computes $V_{\mathsf{P_R}} := \sum_{i=1}^n \chi_i \cdot v_i - y$. They send $V_{\mathsf{P_S}}, V_{\mathsf{P_R}}$ to $\mathcal{F}_{\mathsf{EQ}}$. If it returns (abort), then abort.
9. $\mathsf{P_S}$ outputs $(\mathbf{u}, \mathbf{w})$, and $\mathsf{P_R}$ outputs $\mathbf{v}$.

[a] Instead of sending the whole vector $\boldsymbol{\xi}$, $\mathsf{P_S}$ can send a κ bit random seed which is then expanded with a PRG to obtain $\boldsymbol{\xi}$.
[b] Again, $\mathsf{P_S}$ can send a short seed instead of $\boldsymbol{\chi}$.

Fig. 5. Protocol instantiating $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ in the $(\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}, \mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model.

Definition 3. *We say that a function* $f = (f_0, f_1)\colon \{0,1\}^\kappa \to \mathbb{Z}_{2^\ell} \times \mathbb{F}_{2^{\sigma'}}, x \mapsto (f_0(x), f_1(x))$ *is* right-half injective*, if its restriction to the right-half of the output space* $f_1\colon \{0,1\}^\kappa \to \mathbb{F}_{2^{\sigma'}}$ *is injective.*

In order to achieve active security of our construction, we provide an additional algorithm Check, together with a finite challenge set Ξ. This algorithm, on input $n, \alpha, k\{\alpha\}$, a challenge ξ and a checking value Γ outputs $\top$ or $\bot$.

Definition 4 (PPRF consistency). *Let* F *be a PPRF and let* Ξ *be a challenge set whose size depends on a statistical security parameter* σ*. Consider the following game for* Check*:*

1. $(k\{1\}, \ldots, k\{n\}, \mathsf{state}) \leftarrow \mathcal{A}(1^\kappa, n)$.
2. $\xi \in_R \Xi$
3. $\Gamma \leftarrow \mathcal{A}(1^\kappa, \mathsf{state}, \xi)$
4. *For all* $\alpha \in [n]$*, let* $\mathbf{v}^\alpha \leftarrow \mathsf{PuncEval}(1^\kappa, \alpha, k\{\alpha\})$.
5. *Define* $I := \{\alpha \in [n] \mid \top = \mathsf{Check}(n, \alpha, k\{\alpha\}, \xi, \Gamma)\}$.
6. *We say* $\mathcal{A}$ *wins the game if there exists* $\alpha \neq \alpha' \in I$ *such that there is an index* $i \in [n] \setminus \{\alpha, \alpha'\}$ *with* $v_i^\alpha \neq v_i^{\alpha'}$.

We say that F *has* consistency *if no algorithm* $\mathcal{A}$ *wins the above game with probability more than* $2^{-\sigma}$.

Our algorithms $\mathsf{GGM.KeyGen}, \mathsf{GGM.Gen}$, $\mathsf{GGM.PuncEval}$, $\mathsf{GGM.Check}$, which are used to generate the key, set up the punctured keys, evaluate and check consistency of the punctured keys in our protocol are then as follows:

1. $\mathsf{GGM.KeyGen}(1^\kappa)$ samples $k \in \{0,1\}^\kappa$ uniformly at random and outputs it.
2. $\mathsf{GGM.Gen}(n, k)$ where $n = 2^h$ and $k \in \{0,1\}^\kappa$ is a key:
   (a) Set $K_0^0 \leftarrow k$.
   (b) For each level $i \in [h]$, and for $j \in \{0, \ldots, 2^{i-1}-1\}$ compute $(K_{2j}^i, K_{2j+1}^i) \leftarrow \mathsf{G}(K_j^{i-1})$.
   (c) For $i \in [h]$, set $\overline{K}_0^i \leftarrow \bigoplus_{j=0}^{2^{i-1}-1} K_{2j}^i$ and $\overline{K}_1^i \leftarrow \bigoplus_{j=0}^{2^{i-1}-1} K_{2j+1}^i$.
   (d) For $j \in [2^h]$ compute $v_j, t_j \leftarrow \mathsf{G}'(K_{j-1}^h)$, and set $\mathbf{v} := (v_1, \ldots, v_{2^h})$ and $\mathbf{t} := (t_1, \ldots, t_{2^h})$.
   (e) Compute $\overline{K}_1^{h+1} \leftarrow \sum_{j \in [2^h]} t_i$.
   (f) Output $(\mathbf{v}, \mathbf{t}, (\overline{K}_0^i, \overline{K}_1^i)_{i \in [h]}, \overline{K}_1^{h+1})$.
3. $\mathsf{GGM.PuncEval}(n, \alpha, (\overline{K}^i)_{i \in [h+1]})$ where $n = 2^h$, $\alpha \in [n]$, and $\overline{K}^i \in \{0,1\}^\kappa$:
   (a) Set $K_{\overline{\alpha}_1}^1 \leftarrow \overline{K}^1$.
   (b) For each level $i \in \{2, \ldots, h\}$:
      i. Let $x := \sum_{j=1}^{i-1} 2^{j-1} \cdot \alpha_{i-j}$
      ii. For $j \in \{0, \ldots, 2^{i-1}-1\} \setminus \{x\}$, compute $(K_{2j}^i, K_{2j+1}^i) \leftarrow \mathsf{G}(K_j^{i-1})$.
      iii. Compute $K_{2x+\overline{\alpha}_i}^i \leftarrow \overline{K}^i \oplus \bigoplus_{\substack{0 \le j < 2^{i-1} \\ j \neq x}} K_{2j+\overline{\alpha}_i}^i$.
   (c) For the last level $h+1$:
      i. For $j \in [2^h] \setminus \{\alpha\}$ compute $(v_j, t_j) \leftarrow \mathsf{G}'(K_{j-1}^h)$

(d) Output $(v_j)_{j\in[2^h]\setminus\{\alpha\}}$.

4. $\mathsf{GGM.Check}(n, \alpha, (\overline{K}^i)_{i\in[h+1]}, (\xi_i)_{i\in[n]}, \Gamma)$ where $n = 2^h$, and $\overline{K}^i \in \{0,1\}^\kappa$, $\xi_i \in \mathbb{F}_{2^{\sigma'}}$, and $\Gamma \in \mathbb{F}_{2^{\sigma'}}$:
   (a) For $j \in [2^h] \setminus \{\alpha\}$ recompute t_j as in $\mathsf{GGM.PuncEval}$.
   (b) Compute $t_\alpha \leftarrow \overline{K}^{h+1} - \sum_{j\in[2^h]\setminus\{\alpha\}} t_j$.
   (c) If $\Gamma = \sum_{i\in[n]} \xi_i \cdot t_i$, output $\top$. Otherwise, output $\bot$.

In comparison to Definition 2 $\mathsf{GGM.Gen}$ computes a compressed version of all keys. The pseudorandomness for GGM, as defined in Definition 2, follows from the standard pseudorandomness argument of the GGM construction [Kia+13; BW13; BGI14].

The following theorem shows that the check ensures that a corrupted P_1 cannot create an inconsistent GGM tree, where P_2 obtains different values depending on α. We give the proof in Appendix C.

Theorem 5 (Consistency of the GGM Tree). *Let $n = 2^h \in \mathbb{N}$, $\sigma' = \sigma + 2h$, and G, G' as above, and let $\mathcal{A}$ be any time adversary. If G' is right-half injective, then $\mathcal{A}$ can win the game in Definition 4 with probability at most $2^{-(\sigma+1)}$.*

3.2 Security of $\Pi^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$

Theorem 6. *The protocol $\Pi^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ (Figure 5) securely realizes the functionality $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ in the $(\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}, \mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model: No PPT environment $\mathcal{Z}$ can distinguish the real execution of the protocol from a simulated one except with probability $2^{-(\sigma+1)} + \mathsf{negl}(\kappa)$.*

In the proof, we construct simulators for a corrupted sender and receiver. For the corrupted sender, the simulator follows the protocol by behaving like an honest receiver, but additionally extracts α from the interactions of the dishonest sender with $\mathcal{F}_{\mathsf{OT}}$ and β from the VOLE. Its choice of GGM tree as well as other messages are used to define a consistent vector $\mathbf{w}$ that it sends to the functionality. A subtlety here is simulating the equality check in Step 8d of the protocol, as a corrupt sender can pass this with an ill-formed x^* if it can guess a portion of Δ used in the VOLE-functionality correctly. The simulator must make a key query to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ to simulate the success event correctly. Another issue is that d sent by an honest receiver has a different distribution than how it is chosen in the simulation, but we show that any distinguisher can break the pseudorandomness of the GGM PPRF.

In the simulation for the corrupted receiver, the simulator first translates $\mathcal{F}_{\mathsf{OT}}$ inputs into leakage queries to the functionality. For this, we know that due to Step 6c any adversarial choice leads to consistent GGM tree leaves, so the simulator chooses the set of indices where the check in this Step would pass as leakage input to the functionality $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$. This query then allows the simulator to create a valid transcript: if the attacker guessed α exactly correct (the set is of size 1), then the simulator obtains β from the functionality and can directly

follow the protocol with the honest inputs. If the adversary instead guessed a set of size > 1 correctly that contains the secret α, then the simulator can reconstruct the whole GGM tree and thus a potential input $\mathbf{v}$. This furthermore allows the simulator to detect an inconsistent d that is sent by the corrupt receiver. An inconsistent d can be shown to translate into a selective failure attack on the equality check in Step 8d of the protocol, which requires the simulator to make the second leakage query. If it succeeds, then it obtains α and can adjust $\mathbf{v}_\alpha$ accordingly.

The full proof of Theorem 6, together with a summary of the protocol complexity, can be found in Appendix C.

4 Vector OLE Construction

Given our single-point VOLE protocol, we build a protocol for random VOLE extension over $\mathbb{Z}_{2^\ell}$ by running t single-point instances of length n/t, and concatenating their outputs to obtain a weight t VOLE correlation of length n. Then, these (together with some additional VOLEs) can be extended into pseudorandom VOLEs by applying the primal LPN assumption over $\mathbb{Z}_{2^\ell}$ with regular noise vectors of weight t. Since our single-point protocol introduces some leakage on the hidden point, we need to rely on a variant of LPN with some leakage on the regular noise coordinates.

4.1 Leaky Regular LPN Assumption

The assumption, below, translates the leakage from the single-point VOLE functionality (Figure 4) into leakage on the LPN error vector. Note that there are two separate leakage queries: the first of these allows the adversary to try and guess a single predicate on the entire noise vector, and aborts if this guess is incorrect. This is similar to previous works [Boy+19; Wen+21], and essentially only leaks 1 bit of information on average on the position of the non-zero entries. The second query, in Step 5 is more powerful, since for each query made by the adversary, the exact position of one noise coordinate is leaked with probability 1/2. Intuitively, this means that up to c coordinates of the error vector can be leaked with probability 2^{-c}. In Appendix B.3, we discuss how to select parameters such as to mitigate the effect of this leakage.

Definition 7. *Let $\mathbf{A} \leftarrow \mathsf{G}(m, n, 2^\ell) \in \mathbb{Z}_{2^\ell}^{m \times n}$ be a primal-LPN matrix, and consider the following game $G_b(\kappa)$ with a PPT adversary $\mathcal{A}$, parameterized by a bit b and security parameter κ:*

1. *Sample $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_t) \leftarrow \mathbb{Z}_{2^\ell}^n$, where each sub-vector $\mathbf{e}_i \in \mathbb{Z}_{2^\ell}^{n/t}$ has exactly one non-zero entry in $\mathbb{Z}_{2^\ell}^*$, in position α_i, and sample $\mathbf{s} \leftarrow \mathbb{Z}_{2^\ell}^m$ uniformly*
2. *$\mathcal{A}$ sends sets $I_1, \ldots, I_t \subset [n/t]$*
3. *If $\alpha_j \in I_j$ for all $j \in [t]$, send OK to $\mathcal{A}$, otherwise abort. Additionally, for any j where $|I_j| = 1$, send $\mathbf{e}_j$ to $\mathcal{A}$*
4. *$\mathcal{A}$ sends sets $J_1, \ldots, J_t \subset [n/t]$*

5. *For each* J_i *where* $|J_i| = n/(2t)$*: if* $\alpha_i \in J_i$*, send* α_i *to* $\mathcal{A}$*, otherwise abort*
6. *Let* $\mathbf{y}_0 = \mathbf{s} \cdot \mathbf{A} + \mathbf{e}$ *and sample* $\mathbf{y}_1 \leftarrow \mathbb{Z}_{2^\ell}^n$
7. *Send* $\mathbf{y}_b$ *to* $\mathcal{A}$
8. $\mathcal{A}$ *outputs a bit* b' *(if the game aborted, set the output to* $\perp$*)*

The assumption is that $|\Pr[\mathcal{A}^{G_0(\kappa)} = 1] - \Pr[\mathcal{A}^{G_1(\kappa)} = 1]|$ *is negligible in* κ*.*

4.2 Vector OLE Protocol

Our complete VOLE protocol is given in Figure 6. It realises the functionality $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ (Figure 1), which is the same functionality used for base VOLEs in our single-point protocol. This allows us to use the same kind of "bootstrapping" mechanism as [Wen+21], where a portion of the produced VOLE outputs is reserved to be used as the base VOLEs in the next iteration of the protocol.

In the Init phase of the protocol, the parties create a base VOLE of length m, defining the random LPN secret $\mathbf{u}$, given to the sender, and the scalar Δ, given to the receiver. Then, in each call to Extend, the parties run t instances of $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$ to generate $\mathbf{c} = (c_1, \ldots, c_t)$ and $\mathbf{e} = (e_1, \ldots, e_t)$ for the sender and $\mathbf{b} = (b_1, \ldots, b_t)$ for the receiver. The sender then simply computes $\mathbf{x} \leftarrow \mathbf{u} \cdot \mathbf{A} + \mathbf{e} \in \mathbb{Z}_{2^r}^n$ and $\mathbf{z} \leftarrow \mathbf{w} \cdot \mathbf{A} + \mathbf{c} \in \mathbb{Z}_{2^r}^n$ and the receiver computes $\mathbf{y} = \mathbf{v} \cdot \mathbf{A} + \mathbf{b} \in \mathbb{Z}_{2^r}^n$. This results in the sender holding $\mathbf{x}, \mathbf{z}$ and the receiver holding $\mathbf{y}$ such that $\mathbf{z} = \mathbf{x} \cdot \Delta + \mathbf{y}$. The first m entries of these are reserved to define a fresh LPN secret for the next call to Extend, while the remainder are output by the parties.[4]

Theorem 8. *The protocol* $\Pi_{\mathsf{vole2k}}^{\ell,s}$ *in Fig. 6 securely realizes the functionality* $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ *in the* $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$*-hybrid model, under the leaky regular LPN assumption.*

The proof, given in Appendix D, is straightforward for the malicious sender, and for the malicious receiver we translate the protocol into an instance of primal LPN from Definition 1, which yields indistinguishability.

Communication Complexity When we instantiate the single-point VOLE with our protocol $\Pi_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$ from Section 3, use the equality test sketched in Section 2.3, and Silent OT [Boy+19; Yan+20; CRR21], our VOLE extension protocol $\Pi_{\mathsf{vole2k}}^{\ell,s}$ with LPN parameters, (m, t, n) requires $m + 2t$ base VOLEs and $4t\ell + 2t\sigma + 4t\lceil \log n/t \rceil + (5 + 2\lceil \log n/t \rceil)t\kappa$ bit of communication. The costs for the single-point VOLE protocol are broken down in Appendix C.3.

5 QuarkSilver: QuickSilver Modulo 2^k

We now construct the QuarkSilver zero-knowledge proof system, which is based on a similar principle as the QuickSilver protocol. The main technique to achieve soundness in QuickSilver [Yan+21], similar to LPZK [DIO21], is that a dishonest

[4] In our implementation, we actually reserve $m + 2t$ of the outputs, since we need 2 extra VOLEs for each execution of the protocol for $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$.

VOLE for $\mathbb{Z}_{2^k}$: $\Pi^{\ell,s}_{\mathsf{vole2k}}$

Parameters Fix some parameters:
- n: LPN output size
- m: LPN secret size
- t: number of error coordinates for LPN (assume that $t \mid n$)
- n/t: size of a block in regular LPN
- $\mathbf{A} \in \mathbb{Z}_{2^\ell}^{m \times n}$ is the generator matrix used in primal-LPN

Init This must be called by the parties first and is executed once.
1. $\mathsf{P_S}$ and $\mathsf{P_R}$ send (Init) to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$, and $\mathsf{P_R}$ receives $\Delta \in \mathbb{Z}_{2^s}$.
2. $\mathsf{P_S}$ and $\mathsf{P_R}$ send (Extend, m) to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$. $\mathsf{P_S}$ receives $\mathbf{u}, \mathbf{w} \in \mathbb{Z}_{2^\ell}^m$, and $\mathsf{P_R}$ receives $\mathbf{v} \in \mathbb{Z}_{2^\ell}^m$, such that $\mathbf{w} = \Delta \cdot \mathbf{u} + \mathbf{v}$ over $\mathbb{Z}_{2^\ell}$.

Extend This protocol can be executed multiple times.
1. For $i \in [t]$, $\mathsf{P_S}$ and $\mathsf{P_R}$ send $(\mathsf{SP\text{-}Extend}, n/t)$ to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ which returns $\mathbf{e}_i, \mathbf{c}_i$ to $\mathsf{P_S}$ and $\mathbf{b}_i$ to $\mathsf{P_R}$ such that $\mathbf{c}_i = \Delta \cdot \mathbf{e}_i + \mathbf{b}_i$ over $\mathbb{Z}_{2^\ell}^{n/t}$, and $\mathbf{e}_i \in \mathbb{Z}_{2^\ell}^{n/t}$ has exactly one entry invertible modulo 2^ℓ and zeros everywhere else.
2. Define $\mathbf{e} := (\mathbf{e}_1, \ldots, \mathbf{e}_t) \in \mathbb{Z}_{2^\ell}^n$, $\mathbf{c} := (\mathbf{c}_1, \ldots, \mathbf{c}_t) \in \mathbb{Z}_{2^\ell}^n$, and $\mathbf{b} := (\mathbf{b}_1, \ldots, \mathbf{b}_t) \in \mathbb{Z}_{2^\ell}^n$. Then $\mathsf{P_S}$ computes $\mathbf{x} := \mathbf{u} \cdot \mathbf{A} + \mathbf{e} \in \mathbb{Z}_{2^\ell}^n$, and $\mathbf{z} := \mathbf{w} \cdot \mathbf{A} + \mathbf{c} \in \mathbb{Z}_{2^\ell}^n$. $\mathsf{P_R}$ computes $\mathbf{y} := \mathbf{v} \cdot \mathbf{A} + \mathbf{b} \in \mathbb{Z}_{2^\ell}^n$.
3. $\mathsf{P_S}$ updates $\mathbf{u}, \mathbf{w}$ by setting $\mathbf{u} := \mathbf{x}[0 : m) \in \mathbb{Z}_{2^\ell}^m$ and $\mathbf{w} := \mathbf{z}[0 : m) \in \mathbb{Z}_{2^\ell}^m$, and outputs $(\mathbf{x}[m : n), \mathbf{z}[m : n)) \in \mathbb{Z}_{2^\ell}^\ell \times \mathbb{Z}_{2^\ell}^\ell$. $\mathsf{P_R}$ updates $\mathbf{v}$ by setting $\mathbf{v} := \mathbf{y}[0 : m) \in \mathbb{Z}_{2^\ell}^m$ and outputs $\mathbf{y}[m : n) \in \mathbb{Z}_{2^\ell}^\ell$.

Fig. 6. Protocol for VOLE over $\mathbb{Z}_{2^k}$ in the $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$-hybrid model. Based on [Wen+21].

prover can only cheat in multiplication checks if it can come up with a quadratic polynomial of a certain form, which has a root Δ unknown to the prover. This is straightforward over fields, but over $\mathbb{Z}_{2^k}$ there might be many more than just two roots for a polynomial. Before constructing the zero-knowledge protocol, we therefore give upper-bounds on the number of roots of certain quadratic polynomials over $\mathbb{Z}_{2^k}$.

5.1 Bounded Solutions to Quadratic Equations in $\mathbb{Z}_{2^k}$

We examine the roots of the following polynomial modulo 2^ℓ:

$$f(x) = ax^2 + bx + c$$

We are only interested in the case where $a \neq 0 \mod 2^k$, while b and c may be chosen arbitrarily by the adversary. Finally, we also only look at roots $x \in \{0, \ldots, 2^s - 1\}$, since the secret MAC key Δ is sampled from this range.

Towards giving a bound, we will use the following basic fact about modular square roots.

Proposition 9. *Let $a \in \mathbb{Z}$ be an odd number. Then $x^2 = a \mod 2^\ell$ has at most 4 solutions.*

The proof is standard and can be found in Appendix E.1.

We also use a version of Hensel's lemma (see e.g. [Hac07]), which allows lifting certain solutions to an equation modulo p up to solutions modulo p^ℓ.

Lemma 10 (Hensel's lemma). *Let p be prime, $f(x)$ be a polynomial with integer coefficients and $f'(x)$ its derivative. If there exists an integer x^* such that*

$$f(x^*) = 0 \mod p^i \quad \textit{and} \quad f'(x^*) \neq 0 \mod p$$

then there is a unique integer y modulo p^{i+1} satisfying

$$f(y) = 0 \mod p^{i+1} \quad \textit{and} \quad x^* = y \mod p^i$$

Note that any solution to $f(x) = 0$ modulo p^ℓ is also a solution modulo p. Hence, if the derivatives of all the roots modulo p are non-zero, we are guaranteed that there are no more than two solutions modulo higher powers. The challenging case is when the derivative is zero. We now show the following.

Lemma 11. *Let $f(x) \in \mathbb{Z}[x]$ be a quadratic equation such that 2^r is the largest power of 2 dividing all coefficients. Then for any $\ell, s, s' \in \mathbb{N}$ such that $\ell - r > s'$ there are at most $2^{\max\{(2s-s')/2,1\}}$ solutions to $f(x) = 0 \mod 2^\ell$ in $\{0, \ldots, 2^s - 1\}$.*

Proof. First, we will divide $f(x)$ by 2^r, the largest power of two that divides all coefficients, then redefine f accordingly and solve:

$$f(x) = ax^2 + bx + c = 0 \mod 2^{\ell-r}$$

where now at least one of $\{a, b, c\}$ is odd.

Case 1: a and b are odd. We can use Lemma 10, since the derivative $f'(x) = 2ax+b$ is odd and, therefore, non-zero modulo 2. This means any solution modulo 2 lifts to a unique solution modulo higher powers, so there are at most 2 solutions modulo $2^{\ell-r}$.

Case 2: a is odd and b is even. Since a is invertible modulo $2^{\ell-r}$, we can complete the square: Define $g(y) := a \cdot y^2 + t$ with $t := c - b^2/4 \cdot (a^{-1} \bmod 2^{\ell-r})$. Then we have $f(x) = g(y) \pmod{2^{\ell-r}}$ using the substitution $y = x + b/2 \cdot (a^{-1} \bmod 2^{\ell-r})$:

$$\begin{aligned} g(y) &= a \cdot \left(x + b/2 \cdot (a^{-1} \bmod 2^{\ell-r})\right)^2 + c - b^2/4 \cdot (a^{-1} \bmod 2^{\ell-r}) \\ &= a \cdot x^2 + b \cdot x + b^2/4 \cdot (a^{-1} \bmod 2^{\ell-r}) + c - b^2/4 \cdot (a^{-1} \bmod 2^{\ell-r}) \\ &= ax^2 + bx + c = f(x) \pmod{2^{\ell-r}} \end{aligned}$$

So, to solve for x we can now solve $ay^2 = -t \mod 2^{\ell-r}$ for y, where the original constraint for x now puts y in the interval $\{b/2 \cdot (a^{-1} \bmod 2^{\ell-r}), \ldots, b/2 \cdot (a^{-1} \bmod 2^{\ell-r}) + 2^s - 1\}$. Since this substitution is just a constant shift, the maximal number of possible solutions in any interval of length 2^s for g directly translates into an upper bound on the number of solutions for f in the solution space.

Letting $t' = t \cdot (a^{-1} \bmod 2^{\ell-r})$, we now want to solve:

$$y^2 = -t' \pmod{2^{\ell-r}} \tag{1}$$

- **Case (2a):** $t' = 0 \pmod{2^{\ell-r}}$**.** Then, the solutions y are all the multiples of $2^{(\ell-r)/2}$. In an interval of length 2^s, there can be at most $2^s/2^{(\ell-r)/2} < 2^{s-s'/2}$ of these.
- **Case (2b):** $t' \neq 0 \pmod{2^{\ell-r}}$**.** Let $2^{v'}$ be the largest power of two dividing t'. Since $-t'$ is a square, v' must be even so we can write $v' = 2v$ for some $v \leq (\ell - r - 1)/2$. Write $-t' = u \cdot 2^{2v}$ for some odd $u \in \mathbb{Z}$, and let $z = y/2^v$, so we have

$$z^2 = u \pmod{2^{\ell-r-2v}}. \tag{2}$$

Any solution y for (1) satisfies $y = z \cdot 2^v \bmod 2^{\ell-r-2v}$ for some z that is a solution to (2). So, there is a $k \in \mathbb{Z}$ such that

$$\begin{aligned} & y = z \cdot 2^v + k \cdot 2^{\ell-r-2v} \\ \Rightarrow\ & y^2 = z^2 \cdot 2^{2v} + 2 \cdot z \cdot k \cdot 2^{\ell-r-v} + k^2 \cdot 2^{2(\ell-r-2v)} \\ \Rightarrow\ & y^2 = -t' + z \cdot k \cdot 2^{\ell-r-v+1} \pmod{2^{\ell-r}}. \end{aligned} \tag{3}$$

To bound the number of solutions y, it suffices to bound the number of z and k satisfying the above. For the y's to be distinct mod $2^{\ell-r}$, we need $k < 2^{2v}$, which means there are 2^{v+1} possibilities for k, given by $k = i \cdot 2^{v-1}$ for all $i \in \{0, \ldots, 2^{v+1} - 1\}$. Since z is odd, from Proposition 9 there are no more than 4 solutions to (2), which are of the form $\pm z_0$ and $2^{\ell-r-2v-1} \pm z_0$ for some z_0. However, it is easy to see that plugging $z := z_0 + 2^{\ell-r-2v-1}$ into (3) gives the same set of solutions for y as with $z := z_0$, so we only need to count $\pm z_0$. Overall, this shows there are at most 2^{v+2} solutions y to (3).
Since each solution in the set defined in Equation (3) (with $k = i \cdot 2^{v-1}$) is spaced apart by $2^{\ell-r-v-1}$, any interval of size 2^s contains no more than $2^{s-\ell+r+v+1}$ of these. From the fact that $v \leq (\ell - r - 1)/2$, we get

$$2^{s-\ell+r+v+1} \leq 2^{(\ell-r-1)/2-\ell+r+s+1)} = 2^{(-(\ell-r)+1+2s)/2} \leq 2^{(2s-s')/2}$$

as required (where the last inequality holds since $\ell - r > s'$).

Case 3: a is even, b is odd. In this case, $f(x) = x + c \mod 2$ is linear, hence, the unique solution $x = c \bmod 2$ gives a unique solution modulo $2^{\ell-r}$ via Lemma 10.

Case 4: a, b are even, c is odd. Here, $f(x) = 0$ has no solutions modulo 2, so also no solutions modulo any higher power. □

5.2 Bounded Solutions for a Generalized Setting

In the previous subsection, we analyzed the setting that one would end up with when constructing a soundness argument for our check for one multiplication. In order to amortize this check to t multiplications, we generalize the security game in the following theorem.

Theorem 12. *Let $\ell, s, k \in \mathbb{N}^+$ so that $\ell \geq k + 2s$ and consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

1. *$\mathcal{C}$ chooses $\Delta \in \mathbb{Z}_{2^s}$ uniformly at random.*
2. *$\mathcal{A}$ sends $\delta_0, \ldots, \delta_t \in \mathbb{Z}$ such that not all δ_i for $i > 0$ are $0 \bmod 2^k$.*
3. *$\mathcal{C}$ chooses $\chi_1, \ldots, \chi_t \leftarrow \mathbb{Z}_{2^s}$ uniformly at random and sends these to $\mathcal{A}$.*
4. *$\mathcal{A}$ sends $b, c \in \mathbb{Z}$.*
5. *$\mathcal{A}$ wins iff $(\delta_0 + \sum_i \chi_i \delta_i)\Delta^2 + b\Delta + c = 0 \bmod 2^\ell$.*

Then $\mathcal{A}$ can win with probability at most $(\ell - k + 2) \cdot 2^{-s+1}$.

The proof of Theorem 12 follows a similar way as Lemma 1 of [Cra+18]. The key observation is that Step 3 determines an upper-bound on r, the largest number such that 2^r divides all coefficients of the polynomial. This is because no choice of b, c can increase r as it also must divide the leading coefficient, which is randomized. By the random choice of the χ_i, one can show that the larger r is, the smaller the chance that it divides $\delta_0 + \sum_i \chi_i \delta_i$.

Since a larger r leads to more roots of the polynomial, we can then bound the overall attack success for each possible r. The full proof can be found in Appendix E.2, and in Appendix E.3 we show the following corollary.

Corollary 13. *Let $\sigma \geq 7$ be a statistical security parameter. By setting $s := \sigma + \log \sigma + 3$ and $\ell := k + 2s$, any adversary $\mathcal{A}$ can win the game from Theorem 12 with probability at most $2^{-\sigma}$.*

5.3 QuarkSilver

We now construct the QuarkSilver zero-knowledge proof system. Its main building block are linearly homomorphic commitments instantiated from VOLEs over $\mathbb{Z}_{2^\ell}$.

Linearly Homomorphic Commitments. As in the A2B [Bau+21a] zero-knowledge protocols, we use linearly homomorphic commitments from VOLE to authenticate values in $\mathbb{Z}_{2^k}$: Define a commitment $[x]$ to a value $x \in \mathbb{Z}_{2^k}$ known to the prover by a global key $\Delta \in_R \mathbb{Z}_{2^s}$ and values $K[x], M[x] \in_R \mathbb{Z}_{2^\ell}$ with $\ell \geq k + s$ so that

$$K[x] = M[x] + \widetilde{x} \cdot \Delta \pmod{2^\ell} \tag{4}$$

holds for $\widetilde{x} = x \pmod{2^k}$. Here the prover knows $\widetilde{x}$ and $M[x]$, and the verifier knows Δ and $K[x]$. To open the commitment, the prover reveals $\widetilde{x}, K[x]$ to the verifier who checks that the aforementioned equalities hold.

The commitment scheme is linearly homomorphic, as no interaction is needed to compute $[a \cdot x + b]$ from $[x]$ for publicly known $a, b \in \mathbb{Z}_{2^k}$: $\mathcal{P}, \mathcal{V}$ simply update $\widetilde{x}, K[x]$ and $M[x]$ in the appropriate way modulo 2^ℓ. The same linearity also holds when adding commitments. Unfortunately, the upper $\ell - k$ bits of $\widetilde{x}$ may not be uniformly random when opening a commitment. To resolve this, the prover instead opens $[x + 2^k y]$ using a random commitment $[y]$.

How QuarkSilver Works. QuarkSilver follows the established commit-and-prove paradigm for zero-knowledge proofs. For the commitments, we use the linearly homomorphic commitments described above. For a circuit with n inputs and t multiplications, we start by generating $n + t + 2$ authenticated random values $[r_1], \ldots, [r_{n+t+2}]$ with $\widetilde{r}_i \in_R \mathbb{Z}_{2^\ell}$ for $i \in [n + t + 2]$, i.e. commitments to random values. For this, $\mathcal{P}$ and $\mathcal{V}$ call (Extend, $n + t + 2$) to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$. $\mathcal{P}$ then commits to $\mathbf{w}$ using the first n random commitments. Next, the parties evaluate the circuit topologically, computing commitments to the outputs of linear gates using the homomorphism of $[\cdot]$. For each multiplication gate, $\mathcal{P}$ commits to the output using another unused random commitment. It then remains to show that the commitment to the output of the circuit is a commitment to 1 and that all commited outputs of muliplication gates are indeed consistent with the committed inputs.

To verify the committed output wire, QuarkSilver uses the "blinded opening" procedure that was introduced above. This procedure will consume another random commitment. To check validity of a multiplication, observe that for 3 commitments $[w_\alpha], [w_\beta], [w_\gamma]$ with $\gamma = \alpha \cdot \beta \bmod 2^k$ it holds that

$$\underbrace{K[w_\alpha] \cdot K[w_\beta] - \Delta \cdot K[w_\gamma]}_{B} =$$
$$\underbrace{M[w_\alpha] \cdot M[w_\beta]}_{A_0} + \Delta \cdot \underbrace{(\tilde{w}_\alpha \cdot M[w_\beta] + \tilde{w}_\beta \cdot M[w_\alpha] - M[w_\gamma])}_{A_1},$$

where $\mathcal{P}$ can compute A_0, A_1 while $\mathcal{V}$ can compute B. Hence, by sending A_0, A_1 to $\mathcal{V}$ the latter can check that the relation on B, Δ holds. Instead of sending these for every multiplication, we check all t relations simultaneously by having $\mathcal{V}$ choose a string $\chi \leftarrow \mathbb{Z}_{2^s}^t$, so that the prover instead sends $(\sum_i \chi_i A_{0,i}, \sum_i \chi_i A_{1,i})$ while the verifier checks the relation on $\sum_i \chi_i B_i$ and Δ. Since revealing these linear combinations directly might leak information, $\mathcal{P}$ will first blind the opening with the remaining random commitment from the preprocessing.

While the completeness and zero-knowledge of the aforementioned protocol follows directly, we will explain the soundness in more detail in the security proof. The full protocol is presented in Figure 7.

Security of the QuarkSilver Protocol

QuarkSilver Π^k_{QS}

The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ have agreed on a circuit $\mathcal{C}$ over $\mathbb{Z}_{2^k}$ with n inputs and t multiplication gates, and $\mathcal{P}$ holds a witness $\mathbf{w} \in \mathbb{Z}_{2^k}^n$ so that $\mathcal{C}(\mathbf{w}) = 1$.

Preprocessing phase The preprocessing phase is independent of $\mathcal{C}$ and just needs upper bounds on the number of inputs and multiplication gates of $\mathcal{C}$ as input.

1. $\mathcal{P}$ and $\mathcal{V}$ send (Init) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, and $\mathcal{V}$ receives $\Delta \in \mathbb{Z}_{2^s}$.
2. $\mathcal{P}$ and $\mathcal{V}$ send (Extend, n + t + 2) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, which returns authenticated values $([\mu_i])_{i\in[n]}$, $([\nu_i])_{i\in[t]}$, $[o]$, and $[\pi]$, where all $\widetilde{\mu}_i, \widetilde{\nu}_i, \widetilde{o}, \widetilde{\pi} \in_R \mathbb{Z}_{2^\ell}$.

Online phase

1. For each input w_i, $i \in [n]$, $\mathcal{P}$ sends $\delta_i := w_i - \widetilde{\mu}_i$ to $\mathcal{V}$, and both parties locally compute $[w_i] := [\mu_i] + \delta_i$.
2. For each gate $(\alpha, \beta, \gamma, T) \in \mathcal{C}$, in topological order:
   - If $T = \mathsf{Add}$, then $\mathcal{P}$ and $\mathcal{V}$ locally compute $[w_\gamma] := [w_\alpha] + [w_\beta]$.
   - If $T = \mathsf{Mul}$ and this is the ith multiplication gate, then $\mathcal{P}$ sends $d_i := w_\alpha \cdot w_\beta - \widetilde{\nu}_i$, and both parties locally compute $[w_\gamma] := [\nu_i] + d_i$.
3. For the ith multiplication gate, the parties hold $([w_\alpha], [w_\beta], [w_\gamma])$ with $K[w_i] = M[w_i] + \widetilde{w}_i \cdot \Delta$ for $i \in \{\alpha, \beta, \gamma\}$.
   - $\mathcal{P}$ computes $A_{0,i} := M[w_\alpha] \cdot M[w_\beta] \in \mathbb{Z}_{2^\ell}$ and $A_{1,i} := \widetilde{w}_\alpha \cdot M[w_\beta] + \widetilde{w}_\beta \cdot M[w_\alpha] - M[w_\gamma] \in \mathbb{Z}_{2^\ell}$.
   - $\mathcal{V}$ computes $B_i := K[w_\alpha] \cdot K[w_\beta] - \Delta \cdot K[w_\gamma] \in \mathbb{Z}_{2^\ell}$.
4. $\mathcal{P}$ and $\mathcal{V}$ run the following check:
   (a) Set $A_0^* := M[o]$, $A_1^* := \widetilde{o}$, and $B^* := K[o]$ so that $B^* = A_0^* + A_1^* \cdot \Delta$.
   (b) $\mathcal{V}$ samples $\boldsymbol{\chi} \in_R \mathbb{Z}_{2^s}^t$ and sends it to $\mathcal{P}$.
   (c) $\mathcal{P}$ computes $U := \sum_{i\in[t]} \chi_i \cdot A_{0,i} + A_0^* \in \mathbb{Z}_{2^\ell}$ and $V := \sum_{i\in[t]} \chi_i \cdot A_{1,i} + A_1^* \in \mathbb{Z}_{2^\ell}$, and sends (U, V) to $\mathcal{V}$.
   (d) $\mathcal{V}$ computes $W := \sum_{i\in[t]} \chi_i \cdot B_i + B^* \in \mathbb{Z}_{2^\ell}$, and checks that $W = U + V \cdot \Delta \pmod{2^\ell}$. If the check fails, $\mathcal{V}$ outputs false and aborts.
5. For the single output wire w_h, both parties hold $[w_h]$. They first compute $[z] := [w_h] + 2^k \cdot [\pi]$. Then $\mathcal{P}$ sends $\widetilde{z}$ and $M[z]$ to $\mathcal{V}$ who checks that $\widetilde{z} = 1 \pmod{2^k}$ and $K[z] = M[z] + \widetilde{z} \cdot \Delta$. $\mathcal{V}$ outputs true iff the check passes, and false otherwise.

Fig. 7. Zero-knowledge protocol for circuit satisfiability in the $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$-hybrid model with $s := \sigma + \log(\sigma) + 3$ and $\ell := k + 2s$ for statistical security parameter σ.

Theorem 14. *The protocol Π^k_{QS} (Figure 7) securely realizes the functionality $\mathcal{F}^k_{\mathsf{ZK}}$ in the $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$-hybrid model when instantiated with the parameters $s := \sigma +$*

$\log(\sigma)+3$ *and* $\ell := k+2s$*: No unbounded environment* $\mathcal{Z}$ *can distinguish the real execution of the protocol from a simulated one except with probability* $2^{-\sigma+1}$.

As our protocol is an adaption of QuickSilver [Yan+21], the structure of our proof is also similar. The main difference, lies in the proof of soundness of the multiplication check. We will sketch the argument briefly, while the full proof of Theorem 14 can be found in Appendix F.

For the ith multiplication gate (α, β, γ), let $\widetilde{w}_\gamma = \widetilde{w}_\alpha \cdot \widetilde{w}_\beta + e_i \pmod{2^\ell}$, where $\widetilde{w}_\alpha, \widetilde{w}_\beta, \widetilde{w}_\gamma \in \mathbb{Z}_{2^\ell}$ are the committed values in $[w_\alpha], [w_\beta], [w_\gamma]$ and $e_i \in \mathbb{Z}_{2^\ell}$ is a possible error. Suppose that not all $e_i = 0 \pmod{2^k}$ for $i \in [t]$. Then

$$K[w_\gamma] = M[w_\gamma] + \widetilde{w}_\gamma \cdot \Delta = M[w_\gamma] + (\widetilde{w}_\alpha \cdot \widetilde{w}_\beta) \cdot \Delta + e_i \cdot \Delta \pmod{2^\ell}$$

and (also modulo 2^ℓ)

$$\begin{aligned} B_i &= K[w_\alpha] \cdot K[w_\beta] - \Delta \cdot K[w_\gamma] \\ &= (M[w_\alpha] \cdot M[w_\beta]) + (\widetilde{w}_\alpha \cdot M[w_\beta] + M[w_\alpha] \cdot \widetilde{w}_\beta - M[w_\gamma]) \cdot \Delta - e_i \cdot \Delta^2 \\ &= A_{i,0} + A_{i,1} \cdot \Delta - e_i \cdot \Delta^2 \end{aligned}$$

where $A_{i,0}$ and $A_{i,1}$ are as above the values that an honest $\mathcal{P}$ would send. The equations for all gates are aggregated using a random linear combination:

$$\begin{aligned} W &= \sum\nolimits_{i\in[t]} \chi_i \cdot B_i + B^* \\ &= \underbrace{\sum_{i\in[t]} \chi_i \cdot A_{i,0} + A_0^*}_{U} + \underbrace{(\sum_{i\in[t]} \chi_i \cdot A_{i,1} + A_1^*)}_{V} \cdot \Delta - (\sum_{i\in[t]} \chi_i \cdot e_i) \cdot \Delta^2 \end{aligned} \tag{5}$$

Here, U, V denote the values that an honest $\mathcal{P}$ would send. The corrupted $\mathcal{P}^*$ may choose to send $U' := U + e_U$ and $V' := V + e_V$ instead, and $\mathcal{V}$ accepts if $W = U' + V' \cdot \Delta$ holds. Rearranging Equation 5, we get that $\mathcal{V}$ accepts if

$$0 = e_U + e_V \cdot \Delta + \left(\sum\nolimits_{i\in[t]} \chi_i \cdot e_i\right) \cdot \Delta^2 \pmod{2^\ell} \tag{6}$$

holds. The key observation is that the steps in the protocol correspond exactly to the game defined in Theorem 12 and the dishonest prover wins the game, i.e., cheats successfully, if Equation (6) holds. By Corollary 13 the probability that this happens is at most $2^{-\sigma}$.

General Degree-2 Checks. Yang et al. [Yan+21] also provide zero-knowledge proofs for sets of t polynomials of degree d in n variables (in total), where the communication consists of $n+d$ field element – independent of t. With the results proved in Section 5.2, we can directly instantiate this protocol with $d = 2$. This allows us to verify arbitrary degree-2 relations including the important use case of inner products. Extending the check for higher-degree relations is principally possible. However, the number of roots of the corresponding polynomials grows exponentially with increasing degree. Hence, to achieve the same soundness, we would need to increase the ring size further, which reduces the efficiency. We give the full protocol and its security proof in Appendix G.

6 Experiments

In this section we report on the performance of our VOLE protocol $\Pi^{r,s}_{\mathsf{vole2k}}$ (Section 4) and our zero-knowledge proof system QuarkSilver (Section 5). We implemented the protocols in the Rust programming language using the *swanky* framework[5]. Our implementation is open source and available on GitHub under `https://github.com/AarhusCrypto/Mozzarella`.

Our implementation is generic, it allows to plugin any ring type that implements certain interfaces. We implement $\mathbb{Z}_{2^\ell}$ based on 64, 128, 192 and 256 bit integers. Depending on the size of ℓ, we choose the smallest of these types. Hence, running the protocol with, e.g., $\ell = 129$ and $\ell = 192$ has exactly the same computational and communication costs. In our experiments, we choose one representative ring for each considered size. It is possible to further optimize the communication cost of the implementation by transmitting exactly ℓ bits instead of the complete underlying integer value at the additional cost for the (un)packing operations.

6.1 Benchmarking Environment

All benchmarks were run on two servers with Intel Core i9-7960X processors that have 16 cores and 32 threads. Each server has 128 GiB memory available. They are connected via 10 Gigabit Ethernet with an average RTT of 0.25 ms.

We consider different network settings: For the *LAN* setting, we use the network as described above without further restrictions. To emulate a *WAN* setting, we configure Traffic Control in the Linux kernel via the `tc (8)` tool to artificially restrict the bandwidth to 100 Mbit/s, and increase the RTT to 100 ms. Finally, to explore the bandwidth dependence of our VOLE protocol, we consider a set of network settings with 20, 50, 100 and 500 Mbit/s as well as 1 and 10 Gbit/s bandwidth, and an RTT of 1 ms.

6.2 VOLE Experiments

In this section, we evaluate the performance of our VOLE protocol $\Pi^{\ell,s}_{\mathsf{vole2k}}$ (Section 4). We consider the setting of batch-wise VOLE extension: Given set of n_b base VOLEs, we use our protocols to expand them to $n_o + n_b$ VOLEs to obtain a batch of n_o VOLEs plus n_b VOLEs that can be used as base VOLEs to generate the next batch. We do not consider here how the initial set of base VOLEs are created. As performance measure we use the run-time and communication per generated VOLE correlation in one iteration of the protocol.

LPN Parameter Selection. For a triple of LPN parameters (m, t, n), our protocol extends $n_b = m + 2 \cdot t$ base VOLEs to n new ones. Hence, for a target batch size n_o, we need to find (m, t, n) such that $n \geq n_o + n_b$ and the corresponding LPN problem is still considered infeasible w.r.t. the security parameters.

[5] swanky: `https://github.com/GaloisInc/swanky`

As suggested in prior work [Sch+19; Yan+20; Wen+21], we pick the public LPN matrix $\mathbf{A} \in \mathbb{Z}_{2^\ell}^{m \times n}$ as a generator of a 10-local linear code (i.e. each column of $\mathbf{A}$ contains exactly 10 uniform non-zero entries). As discussed in Section 2.5, each non-zero entry is picked randomly from $\mathbb{Z}_{2^\ell}^*$ (i.e. odd), to ensure that reduction modulo 2 does not reduce sparsity. This results in fast computation of the expansion $\mathbf{u} \cdot \mathbf{A}$ (for some $\mathbf{u} \in \mathbb{Z}_{2^\ell}$), as each entry involves only 10 positions of $\mathbf{u}$. We then pick (m, t, n) such that all known attacks on the LPN problem require at least 2^κ operations [Boy+19; Wen+21] (see also Appendix B.2). Note that, as our variant of the regular LPN assumption (Definition 7) leaks blocks of the noise vector, we must pick t such that our protocols are secure in advent of leaking up to $\sigma \in \{40, 80\}$ blocks. To do this, we assume that leaking the noisy index within a single block of $\Pi^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ directly gives an index of the secret and then subtract the leaked block from the noise vector as well as the corresponding index from the secret and make sure that the new problem is still infeasible to solve.

For a given n_o we experimentally find the LPN parameter set (m, t, n) that gives us the best performance while satisfying the above conditions.

We chose LPN parameters targeting a level of $\kappa = 128$ bits of computational security, and used the approach of Boyle et al. [Boy+18] to estimate the hardness of the LPN problem. Recently, Liu et al. [Liu+22] noted that this significantly underestimates the hardness of the LPN problem. Using their estimation, our parameters yield about 153–158 bits of security. Hence, we could reduce the parameters to get a more efficient instantiation of our protocol. We chose to use LPN with odd noise values in $\mathbb{Z}_{2^k}$ to resist the reduction attack of Liu et al. [Liu+22], which otherwise reduces the effective noise rate by half. In case of a potential future attack on LPN with odd noise, with the same impact, we would still achieve 103–109 bits of security.

For more details regarding the choice of LPN parameters and how we estimate the hardness of the leaky LPN problem, we refer to Appendix B.3.

General Benchmarks. For each statistical security level $\sigma \in \{40, 80\}$, we selected two LPN parameter sets (m, t, n) targeting VOLE batch sizes of $n_o \in \{10^7, 10^8\}$. We execute the protocol in two different network settings with four different ring sizes $\ell \in \{64, 104, 144, 244\}$ (one representative for each of the underlying integer types) for each of the parameter sets. Table 1 contains the results of our experiments.

With increasing ring size ℓ the costs increase as the arithmetic becomes more costly and more data needs to be transferred. Moreover, with a larger batch size the costs per VOLE decrease. In terms of run-time and communication costs, it is more efficient to generate a larger amount of VOLEs at once. However, the required resources, e.g., memory consumption, also increase with the batch size. In the WAN setting, a larger batch size is especially more efficient, since the effect of the higher latency is less pronounced on the amortized run-times.

Although the chosen LPN parameter sets worked well in our case, other combinations of m and t can yield a similar performance with same security,

while influencing the computation and communication cost slightly. Such an effect can be noticed in the first parameter sets, where the communication cost decreases when going from $\sigma = 40$ to $\sigma = 80$. It is a trade-off, and we deem experimental verification necessary to choose the best-performing parameter set.

Table 1. Benchmark results of our VOLE protocol. We measure the run-time of the Extend operation in ns per VOLE and the communication cost in bit per VOLE. The benchmarks are parametrized by the ring size ℓ (i.e., using $\mathbb{Z}_{2^\ell}$). The computational security parameter is set to $\kappa = 128$. For statistical security $\sigma \in \{40, 80\}$, we target batch sizes of $n_o = 10^7$ and $n_o = 10^8$, and use the stated LPN parameters (m, t, n).

σ	ℓ	Run-time		Communication		
		LAN	WAN	$\mathsf{P_S} \to \mathsf{P_R}$	$\mathsf{P_R} \to \mathsf{P_S}$	total
40	$m = 553\,600$, $t = 2\,186$, $n = 10\,558\,380$					
	64	27.3	190.8	0.467	0.927	1.394
	104	40.7	186.7	0.509	0.955	1.464
	144	55.2	212.6	0.551	0.983	1.534
	244	80.7	255.0	0.593	1.011	1.604
	$m = 773\,200$, $t = 15\,045$, $n = 100\,816\,545$					
	64	20.1	46.0	0.318	0.636	0.954
	104	33.2	58.9	0.347	0.655	1.002
	144	46.7	75.1	0.376	0.674	1.050
	244	76.7	102.8	0.405	0.694	1.098
80	$m = 830\,800$, $t = 2\,013$, $n = 10\,835\,979$					
	64	27.6	171.9	0.431	0.853	1.284
	104	42.6	194.1	0.469	0.879	1.349
	144	59.4	217.1	0.508	0.905	1.413
	244	89.3	277.4	0.547	0.931	1.477
	$m = 866\,800$, $t = 18\,114$, $n = 100\,913\,094$					
	64	21.4	48.2	0.383	0.765	1.148
	104	34.3	61.0	0.418	0.789	1.206
	144	49.2	76.0	0.453	0.812	1.264
	244	79.8	106.8	0.487	0.835	1.322

Comparison with Wolverine. We compare the efficiency of our VOLE extension protocol with that of Wolverine [Wen+21]. While we use different hardware, we try to replicate their benchmarking setup by restricting our benchmark to maximal 5 threads and up to 64 GiB memory, and select LPN parameters to generate $n_o \approx 10^7$ VOLEs. The results are given in Table 2, where we list our run-times in different bandwidth settings with the corresponding numbers given in [Wen+21]. Note that Wolverine uses the prime field $\mathbb{F}_{2^{61}-1}$, whereas we in-

stantiate our protocol with different larger rings $\mathbb{Z}_{2^\ell}$. In network settings with at least 50 Mbit/s bandwidth, we achieve similar or better performance for the ring sizes up to 128 bit.

Table 2. Run-times in ns per VOLE in different bandwidth settings, when generating ca. 10^7 VOLEs with 5 threads and statistical security $\sigma \geq 40$. The parameter ℓ denotes the size of a ring or field element. The numbers for Wolverine are taken from [Wen+21].

	ℓ	20 Mbit/s	50 Mbit/s	100 Mbit/s	500 Mbit/s	1 Gbit/s	10 Gbit/s
this work	64	110.0	68.7	55.0	50.2	50.6	50.4
	104	142.0	95.2	80.1	73.2	71.5	73.6
	144	178.6	134.7	119.3	111.6	112.6	113.3
	244	266.3	219.1	201.7	194.5	193.7	196.5
Wolverine	61	101.0	87.0	85.0	85.0	85.0	—

Bandwidth Dependence. Table 2 also shows how the available bandwidth affects the performance of our protocol. We observe that increasing the network bandwidth beyond 100 Mbit/s does not improve the run-time significantly. This indicates that the required computation is the bottleneck above this point.

6.3 Zero-Knowledge Experiments

We explore at what rate our QuarkSilver protocol (Section 5) is able to verify the correctness of multiplications. In our experiments we check for $N \approx 10^7$ triples of the form $([w_{i,\alpha}], [w_{i,\beta}], [w_{i,\gamma}])$ for $i \in [N]$ that $w_{i,\alpha} \cdot w_{i,\beta} = w_{i,\gamma} \pmod{2^k}$ holds. Assuming the prover has already committed to $2N$ values $([w_{i,\alpha}], [w_{i,\beta}])$, we execute the following three steps:

1. vole: Perform the Extend operation of $\Pi^{s,\ell}_{\mathsf{vole2k}}$ to create the necessary amount of VOLEs (at least $N + 1$).
2. mult: Step 2 of Π^{k}_{QS} (Figure 7) to commit to the results $w_{i,\gamma} := w_{i,\alpha} \cdot w_{i,\beta}$ of the multiplications.
3. check: Steps 3 and 4 of Π^{k}_{QS} to verify that the multiplications are correct modulo 2^k.

While the execution of $\Pi^{s,\ell}_{\mathsf{vole2k}}$ in Step 1 is parallelized, the further steps are executed in a single thread, and there is still room for optimizations, e.g., using smaller integers for the coefficients of the random linear combination and better interleaving computation and communication.

For statistical security levels of $\sigma = 40$ and $\sigma = 80$, we run the protocol with ring sizes $\ell = 162$ and $\ell = 244$, respectively. This corresponds to the required ring size ℓ to enable zero-knowledge proof over $\mathbb{Z}_{2^k}$ with $k = 64$. It also covers

the $k = 32$ setting, since the corresponding rings (with $\ell \in \{130, 212\}$) are implemented in the same way.

In Table 3 we list the achieved run-times and communication costs per multiplication and show how they are distributed over the three steps of the protocol. We clearly see that the costs are dominated by Step 2, where the majority of the communication happens (one $\mathbb{Z}_{2^\ell}$ element per multiplication). Additional benchmarks show that increasing the bandwidth to more than 500 Mbit/s does not increase the performance.

Table 3. Benchmark results of our QuarkSilver protocol. We measure the run-time of a batch of $\approx 10^7$ multiplications and their verification in ns per multiplication and the communication cost in bit per multiplication. The benchmarks are parametrized by the statistical security parameter σ, and the computational security parameter is set to $\kappa = 128$. For $\sigma = 40$, we use the ring of size $\ell = 162$, for $\sigma = 80$, we use $\ell = 244$.

σ		Run-time		Communication		
		LAN	WAN	$\mathsf{P_S} \to \mathsf{P_R}$	$\mathsf{P_R} \to \mathsf{P_S}$	total
40	vole	78.5	265.5	0.5	1.0	1.5
	mult	663.2	2 101.5	192.0	0.0	192.0
	check	28.2	38.2	0.0	0.0	0.0
	total	769.9	2 405.2	192.5	1.0	193.5
80	vole	125.3	345.6	0.5	0.9	1.5
	mult	680.7	2 767.2	256.0	0.0	256.0
	check	42.3	52.4	0.0	0.0	0.0
	total	848.3	3 165.2	256.5	0.9	257.5

With a completely single-threaded implementation (including single-threaded VOLEs), we can verify about 0.9 million multiplications per second for statistical security parameter $\sigma = 40$ and ring $\mathbb{Z}_{2^{162}}$, compared to (single-threaded) QuickSilver's up to 4.8 million multiplications per second over the field $\mathbb{F}_{2^{61}-1}$, as reported by Yang et al. [Yan+21]. This is a factor 5.3 difference.

When looking at the performance of $\mathbb{Z}_{2^{162}}$ compared to $\mathbb{F}_{2^{61}-1}$, we see that $\mathbb{Z}_{2^{162}}$ ring elements are represented by three 64 bit integers compared to $\mathbb{F}_{2^{61}-1}$ field elements which fit into a single integer. While this results in $3\times$ more communication, the computational costs are also higher: In microbenchmarks, arithmetic operations in $\mathbb{Z}_{2^{162}}$ are $2.1-2.5\times$ slower compared to the corresponding operations in $\mathbb{F}_{2^{61}-1}$ (e.g., $\mathbb{Z}_{2^{162}}$ multiplications require 6 IMUL/MULX instructions, $\mathbb{F}_{2^{61}-1}$ multiplications need one MULX instruction). Moreover, the compiler can automatically vectorize element-wise computations on vectors of field elements with AVX instruction due to the smaller element size, but this is (at least currently) not possible with the larger ring. Computation on rings also results in a slightly higher rate of cache misses, which we attribute to the fact

that more field elements than ring elements fit in a cache line, simply due to their size.

We want to stress that this direct comparison is not necessarily fair, though: The Mersenne prime modulus $p = 2^{61} - 1$ has been chosen because it allows to implement the field arithmetic very efficiently. The plaintext space has roughly the same size in both settings (64 vs. 61 bit), but the arithmetic on the secrets is entirely different which is the main difference of our work to the field-based approach of QuickSilver. While QuarkSilver supports 64 bit arithmetic natively (which is one of the main points of considering $\mathbb{Z}_{2^k}$ protocols), things are more complicated with fields. To emulate 64 bit arithmetic in a prime field, the prime modulus has to have size ≥ 128 bit (so no modular wraparound occurs during multiplications) which means more communication and more complicated arithmetic. Then, one also has to commit to the correct reduction modulo 2^{64} and prove that the reduction is computed correctly, e.g., with range proofs or using the truncation protocols of Baum et al. [Bau+21a] – both are not cheap, in particular given they are needed for each multiplication mod 2^{64} (and possibly additions, too). Moreover, with a prime modulus of this size one cannot take advantage of a Mersenne prime (the nearest Mersenne primes would be $p = 2^{127} - 1$ (too small) and $p = 2^{521} - 1$ (much larger)) to increase computational efficiency.

Acknowledgements

This work is supported by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No. 803096 (SPEC), the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM), the Independent Research Fund Denmark (DFF) under project number 0165-00107B (C3PO), the Aarhus University Research Foundation, and the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0085. 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 the Defense Advanced Research Projects Agency (DARPA). Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). We thank the ENCRYPTO group at TU Darmstadt for allowing us to use their servers for our experiments.

References

[AG11] S. Arora and R. Ge. "New Algorithms for Learning in Presence of Errors". In: *ICALP 2011, Part I*. July 2011. DOI: 10.1007/978-3-642-22006-7_34.

[Ale03] M. Alekhnovich. "More on Average Case vs Approximation Complexity". In: *44th FOCS*. Oct. 2003. DOI: 10.1109/SFCS.2003.1238204.

[Ame+17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *ACM CCS 2017*. Oct. 2017. DOI: 10.1145/3133956.3134104.

[App+17] B. Applebaum, I. Damgård, Y. Ishai, M. Nielsen, and L. Zichron. "Secure Arithmetic Computation with Constant Computational Overhead". In: *CRYPTO 2017, Part I*. Aug. 2017. DOI: 10.1007/978-3-319-63688-7_8.

[Bau+21a] C. Baum, L. Braun, A. Munch-Hansen, B. Razet, and P. Scholl. "Appenzeller to Brie: Efficient Zero-Knowledge Proofs for Mixed-Mode Arithmetic and Z2k". In: *ACM CCS 2021*. Nov. 2021. DOI: 10.1145/3460120.3484812.

[Bau+21b] C. Baum, A. J. Malozemoff, M. B. Rosen, and P. Scholl. "Mac'n'Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions". In: *CRYPTO 2021, Part IV*. Aug. 2021. DOI: 10.1007/978-3-030-84259-8_4.

[Bau+22] C. Baum, L. Braun, A. Munch-Hansen, and P. Scholl. "Moz$\mathbb{Z}_{2^k}$arella: Efficient Vector-OLE and Zero-Knowledge Proofs Over $\mathbb{Z}_{2^k}$". In: *CRYPTO 2022, Part IV*. Aug. 2022. DOI: 10.1007/978-3-031-15985-5_12.

[Ben+19] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *CRYPTO 2019, Part III*. Aug. 2019. DOI: 10.1007/978-3-030-26954-8_23.

[BGI14] E. Boyle, S. Goldwasser, and I. Ivan. "Functional Signatures and Pseudorandom Functions". In: *PKC 2014*. Mar. 2014. DOI: 10.1007/978-3-642-54631-0_29.

[BKW03] A. Blum, A. Kalai, and H. Wasserman. "Noise-tolerant learning, the parity problem, and the statistical query model". In: *Journal of the ACM (JACM)* 4 (2003).

[Blu+94] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. "Cryptographic Primitives Based on Hard Learning Problems". In: *CRYPTO'93*. Aug. 1994. DOI: 10.1007/3-540-48329-2_24.

[Boy+18] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. "Compressing Vector OLE". In: *ACM CCS 2018*. Oct. 2018. DOI: 10.1145/3243734.3243868.

[Boy+19] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. "Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation". In: *ACM CCS 2019*. Nov. 2019. DOI: 10.1145/3319535.3354255.

[BW13] D. Boneh and B. Waters. "Constrained Pseudorandom Functions and Their Applications". In: *ASIACRYPT 2013, Part II*. Dec. 2013. DOI: 10.1007/978-3-642-42045-0_15.

[Cra+18] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. "SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority". In: *CRYPTO 2018, Part II*. Aug. 2018. DOI: 10.1007/978-3-319-96881-0_26.

[CRR21] G. Couteau, P. Rindal, and S. Raghuraman. "Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes". In: *CRYPTO 2021, Part III*. Aug. 2021. DOI: 10.1007/978-3-030-84252-9_17.

[DIO21] S. Dittmer, Y. Ishai, and R. Ostrovsky. "Line-point zero knowledge and its applications". In: *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*. 2021.

[DT17] T. Debris-Alazard and J.-P. Tillich. "Statistical decoding". In: *2017 IEEE International Symposium on Information Theory (ISIT)*. 2017.

[EKM17] A. Esser, R. Kübler, and A. May. "LPN Decoded". In: *CRYPTO 2017, Part II*. Aug. 2017. DOI: 10.1007/978-3-319-63715-0_17.

[GGM84] O. Goldreich, S. Goldwasser, and S. Micali. "How to Construct Random Functions (Extended Abstract)". In: *25th FOCS*. Oct. 1984. DOI: 10.1109/SFCS.1984.715949.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. "How to construct random functions". In: *Journal of the ACM (JACM)* 4 (1986).

[GNS21] C. Ganesh, A. Nitulescu, and E. Soria-Vazquez. *Rinocchio: SNARKs for Ring Arithmetic*. Cryptology ePrint Archive, Report 2021/322. https://eprint.iacr.org/2021/322. 2021.

[Hac07] P. Hackman. *Elementary Number Theory*. 2007.

[HK97] S. Halevi and H. Krawczyk. "MMH: Software Message Authentication in the Gbit/Second Rates". In: *FSE'97*. Jan. 1997. DOI: 10.1007/BFb0052345.

[Kia+13] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. "Delegatable pseudorandom functions and applications". In: *ACM CCS 2013*. Nov. 2013. DOI: 10.1145/2508859.2516668.

[Liu+22] H. Liu, X. Wang, K. Yang, and Y. Yu. *The Hardness of LPN over Any Integer Ring and Field for PCG Applications*. Cryptology ePrint Archive, Report 2022/712. https://eprint.iacr.org/2022/712. 2022.

[Lyu05] V. Lyubashevsky. "The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem". In: *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. 2005. ISBN: 978-3-540-31874-3.

[Mal+19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings". In: *ACM CCS 2019*. Nov. 2019. DOI: 10.1145/3319535.3339817.

[Pra62] E. Prange. "The use of information sets in decoding cyclic codes". In: *IRE Transactions on Information Theory* 5 (1962). DOI: 10.1109/TIT.1962.1057777.

[Sch+19] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. "Distributed Vector-OLE: Improved Constructions and Implementation". In: *ACM CCS 2019*. Nov. 2019. DOI: 10.1145/3319535.3363228.

[Sch18] P. Scholl. "Extending Oblivious Transfer with Low Communication via Key-Homomorphic PRFs". In: *PKC 2018, Part I*. Mar. 2018. DOI: 10.1007/978-3-319-76578-5_19.

[Wen+21] C. Weng, K. Yang, J. Katz, and X. Wang. "Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits". In: *2021 IEEE Symposium on Security and Privacy*. May 2021. DOI: 10.1109/SP40001.2021.00056.

[Yan+20] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang. "Ferret: Fast Extension for Correlated OT with Small Communication". In: *ACM CCS 2020*. Nov. 2020. DOI: 10.1145/3372297.3417276.

[Yan+21] K. Yang, P. Sarkar, C. Weng, and X. Wang. "QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field". In: *ACM CCS 2021*. Nov. 2021. DOI: 10.1145/3460120.3484556.

[Zic17] L. Zichron. *Locally Computable Arithmetic Pseudorandom Generators*. Master's thesis, School of Electrical Engineering, Tel Aviv University, 2017. http://www.eng.tau.ac.il/~bennyap/pubs/Zichron.pdf. 2017.

A Interesting Ring Sizes for VOLE

Table 4. Overview of the required ring size ℓ and size s of Δ that different zero-knowledge proofs require to verify circuits over $\mathbb{Z}_{2^k}$ with σ bits of statistical security.

σ	k	$\Pi^{\mathbb{Z}_{2^k}}_{\mathsf{ComZK\text{-}a}}$ [Bau+21a]		$\Pi^{\mathbb{Z}_{2^k}}_{\mathsf{ComZK\text{-}b}}$ [Bau+21a]		QuarkSilver (**this work**)	
		$s := \sigma$	$\ell := k + s$	$s := \sigma$	$\ell := k + 2s$	$s := \sigma + \log(\sigma) + 3$	$\ell := k + 2s$
40	32	40	72	40	112	49	130
	64	40	104	40	144	49	162
80	32	80	112	80	192	90	212
	64	80	144	80	224	90	244

B Choosing LPN Parameters

B.1 Dual-LPN

Definition 15 (Dual-LPN). *Let $\mathcal{D}^M_{n,t}$ and G be defined as in Definition 1. Let $n', n \in \mathbb{N}$ be defined so that $n' > n$ and then let $\mathsf{G}^\perp$ be a probabilistic code generation algorithm such that $\mathsf{G}^\perp(n', n, M)$ outputs a matrix in $\{\mathbf{H} \in \mathbb{Z}_M^{n' \times n} \mid \mathrm{rank}(\mathbf{H}) = n \wedge \exists \mathbf{A} \in \mathrm{Im}(\mathsf{G}(n' - n, n', M)) \,.\, \mathbf{A} \cdot \mathbf{H} = 0\}$. Let parameters n, n', t be implicit functions of security parameter κ. The $\mathsf{dual\text{-}LPN}^{\mathsf{G}}_{n',n,t,M}$ assumption states that:*

$$\{(\mathbf{H}, \mathbf{x}) \mid \mathbf{H} \leftarrow \mathsf{G}^\perp(n', n, M), \mathbf{e} \leftarrow \mathcal{D}^M_{n,t}, \mathbf{x} := \mathbf{e} \cdot \mathbf{H}\}$$
$$\approx_C \{(\mathbf{H}, \mathbf{x}) \mid \mathbf{H} \leftarrow \mathsf{G}^\perp(n', n, M), \mathbf{x} \in_R \mathbb{Z}_M^n\}.$$

B.2 Attacks on LPN

We recall the main attacks on LPN, following the analysis of [EKM17; Boy+18; Boy+19]. We refer to [EKM17] for a more thorough overview. Let $\mathcal{D}^M_{n,t}$ be the noise distribution with Hamming weight t. Recall that the LPN secret dimension is m, while the number of samples is n. We define the average noise rate to be $r = t/n$.

Pooled Gaussian Elimination This attack recovers $\mathbf{x}$ from $\mathbf{b} = \mathbf{x} \cdot \mathbf{A} + \mathbf{e}$ by guessing m non-noisy coordinates of $\mathbf{b}$, performs Gaussian elimination to find $\mathbf{x}$ and verifies that the guess of the m non-noisy coordinates was correct. [EKM17] introduced *Pooled* Gaussian elimination in order to reduce the samples required by regular Gaussian elimination. In Pooled Gaussian elimination, the adversary guesses m non-noise samples by picking them at random from a pool of fixed $N = m^2 \log^2 m$ LPN samples in each iteration,

and then inverts the corresponding subsystem to get a potential solution x' and then checks if $x' = x$. For LPN with noise rate r, this attacks recovers the secret in time $\frac{m^3 \log^2 m}{(1-r)^m}$ using $m^2 \log^2 m$ samples.

Information Set Decoding (ISD) [Pra62] Breaking LPN is equivalent to breaking its dual variant, which may be interpreted as the task of decoding a random linear code from its syndrome. The best algorithms for this are improvements of Prange's ISD algorithm, which tries to find a size-t subset of the rows of $\mathbf{B}$ (the parity-check matrix of the code used within the Dual-LPN assumption) that spans $\mathbf{e} \cdot \mathbf{B}$.

The BKW Algorithm [BKW03] This is a variant of Gaussian elimination which achieves subexponential complexity, even for high-noise LPN. It requires a subexponential number of samples and can solve LPN over $\mathbb{F}_2$ in time $2^{O(m/\log(m/r))}$ using $2^{O(m/\log(m/r))}$ samples.

Combinations of the above [EKM17] The authors of [EKM17] conducted an extended study of the security of LPN and they described combinations and refinements of the previously mentioned attack (called *well-pooled Gauss attack*, *hybrid attack* and the *well-pooled MMT attack*). All of these attacks achieve subexponential time complexity, but require as many samples as their time complexity.

Scaled-down BKW [Lyu05] This is a variant of the BKW algorithm, tailored to LPN with polynomially-many samples. It solves the LPN problem in time $2^{O(m/\log\log(m/r))}$, using $m^{1+\epsilon}$ samples (for any $\epsilon > 0$) and has worse performance in time and number of samples for larger fields.

Statistical Decoding [DT17][6] The goal of all of the previous attacks is to recover the secret $\mathbf{x}$, whereas this attack directly attempts to distinguish $\mathbf{b} = \mathbf{x} \cdot \mathbf{A} + \mathbf{e}$ from random. By the singleton bound, the minimal distance of the dual code of $\mathbf{C}$ is at most $m+1$, hence there must be a parity-check equation for $\mathbf{C}$ of weight $m+1$, that is, a vector $\mathbf{v}$ such that $\mathbf{A} \cdot \mathbf{v}^T = 0$. Then, if $\mathbf{b}$ is random, $\mathbf{b} \cdot \mathbf{v}^T = 0$ with probability at most $1/|\mathbb{F}|$, whereas if $\mathbf{b}$ is a noisy encoding, it goes to zero with probability roughly $((n-m-1)/n)^{rn}$ [Boy+18].

Note that some of the above attacks are specialized to LPN over $\mathbb{F}_2$, while others work for more general fields. When working in $\mathbb{Z}_{2^\ell}$, though, one can always reduce the LPN instance modulo 2 and run the distinguisher for the problem in $\mathbb{F}_2$.

B.3 Implications of the Leaky Regular LPN Assumption

We now discuss how the additional leakage in our leaky variant of the LPN assumption (Definition 7) affects the hardness of the problem and the choice of parameters.

The Security Game. Recall the security game from Definition 7 where the adversary $\mathcal{A}$ can make two queries for the indices of the non-zero entries in the error vector $\mathbf{e}$, an I-query (Step 2) and a J-query (Step 4).

[6] Statistical Decoding is also known as Low-Weight Parity Check [App+17; Zic17].

For each query, $\mathcal{A}$ sends a collection of sets $I_1, \ldots, I_t \subseteq [\frac{n}{t}]$ (resp. $J_{i_1}, \ldots, J_{i_h} \subseteq [\frac{n}{t}]$ with $h \in [0, t]$ and all $i_j \in [t]$ different) to the challenger. The adversary then learns whether $\alpha_i \in I_i$ for all $i \in [t]$ (resp. $\alpha_{i_j} \in J_{i_j}$ for all $j \in [h]$) where α_i is the index of the non-zero entry in the subvector $\mathbf{e}_i$. If this is not the case, i.e., there is an index $i \in [t]$ such that $\alpha_i \notin I_i$ (resp. i_j such that $\alpha_{i_j} \notin J_i$), then the game aborts.

The I-query is similar to previous works [Boy+19; Wen+21], and essentially only leaks 1 bit of information on average on the position of the non-zero entries: The adversary then learns whether $\alpha_i \in I_i$ for all $i \in [t]$. In the case that one of the sets contain only the correct index $I_i = \{\alpha_i\}$, our variant additionally reveals the non-zero value β_i. Compared to previous works, the adversary is then able to remove the noise and, thus, learns $\frac{n}{t}$ instead of $\frac{n}{t} - 1$ noiseless equations.

The J-query is more powerful: For each $i \in [t]$, the adversary has the option to make a guess by sending a subset $J_i \subseteq [\frac{n}{t}]$ of size $\frac{n}{2t}$ (sets of other sizes are ignored). In contrast to the I-query, the adversary learns the correct index α_i for each successful guess J_i of this form. If there is a guess J_i of this form such that $\alpha_i \notin J_i$, the game aborts. However, due to the size restriction, every guess independently succeeds with probability $1/2$. Hence, except with probability at most $2^{-\sigma}$, the adversary will not learn more than σ noisy coordinates α_i without without the game aborting.

Estimating Security of Leaky LPN. Let $h(i, j) := (i - 1) \cdot (\frac{n}{t}) + j$ be an index function which computes the index of a length n vector that corresponds to the jth entry of the ith subvector of length $\frac{n}{t}$. We use $\mathbf{A}_k$ to denote the kth column of $\mathbf{A}$.

Suppose the adversary has received an LPN sample $\mathbf{y} = \mathbf{s} \cdot \mathbf{A} + \mathbf{e}$ and learned the position α_i of the noise in block $\mathbf{e}_i$. Then it knows that $y_{h(i,\alpha_i)} = \mathbf{s} \cdot \mathbf{A}_{h(i,\alpha_i)} + \beta_i$ holds for the (unknown) noise value β_i. It also knows that $y_{h(i,j)} = \mathbf{s} \cdot \mathbf{A}_{h(i,j)}$ holds for all other indices in this block $j \in [\frac{n}{t}] \setminus \{\alpha_i\}$, i.e., it now has $\frac{n}{t} - 1$ linear equations of the secret vector $\mathbf{s}$ *without noise*. In the worst case, these could be used to recover $\frac{n}{t} - 1$ entries of $\mathbf{s}$.

Now the adversary can to learn up to σ noisy coordinate except with negligible probability. Therefore, we must tolerate the leakage of up to $\sigma \cdot (\frac{n}{t} - 1)$ noise-free equations or the same number of entries of $\mathbf{s}$. Hence, it can transform the given LPN instance into a smaller instance where the σ affected blocks are removed and the secret is $\sigma \cdot (\frac{n}{t} - 1)$ entries smaller. and we require that it is still infeasible for the adversary to solve this smaller LPN instance.

To summarize, we assume that an instance of the leaky LPN problem with parameters (m, t, n) is as hard as a standard LPN instance (Definition 1) with reduced parameters $(m', t', n') = (m - \sigma \cdot (\frac{n}{t} - 1), t - \sigma, n - \sigma \cdot \frac{n}{t})$. Hence, we must choose (m, t, n) such that n is large enough for our application and that the (standard) regular LPN problem with parameters (m', t', n') is hard to solve.

Estimating Security of Standard LPN. We initially estimated the security of standard regular LPN (Definition 1) with the reduced parameters (m', t', n')

following Boyle et al. [Boy+18] as

$$\log_2 \left(\frac{m'+1}{(1-\frac{m'}{n'})^{t'}} \right) \text{ bits},$$

based on their estimation of the cost of the low-weight parity-check attack.

Recently, Liu et al. [Liu+22] found that the above estimate is very conservative, and the pooled Gauss attack performs better for all practical parameters. They provide a script to compute more precise estimates, and we refer to their work for more details. As mentioned in Section 2.5, by choosing the LPN noise values to be odd, we avoid the reduction attack from Liu et al., which would otherwise halve the noise rate.

Choosing LPN Parameters for VOLE Extension. We start by selecting a number n_o of VOLEs that we want to produce in each iteration. Then we search for parameters (m, t, n) for the leaky LPN problem that gives us $\kappa = 128$ bits of security (see above) such that additionally $n \geq n_o + (m + 2t)$ holds. The latter allows us to generate n_o usable VOLEs in each iteration while keeping $m + 2t$ values to run the next execution of our extension protocol.

In our experiments (Section 6.2), we set $n_o \in \{10^7, 10^8\}$. We then tried different secret sizes m in the interval $[10^5, 10^6]$, computed the required number of noise coordinates t so that the leaky LPN problem is sufficiently hard, and benchmarked our protocol. This gave us parameter sets with the same security properties and output size, out of which we selected the best performing one.

If we fix some values of m and n_o (in the ranges stated above) and compare the required noise coordinates with and without the extra leakage, then we need about $1.5\times$ (resp. $2.1\times$) more noise coordinates to compensate for the leakage for $\sigma = 40$ bits (resp. 80 bits) of statistical security.

For the concrete parameter sets used the in experiments and the achieved security level in context of the discussion above, we refer to Section 6.2.

C Proof of the Single-Point VOLE protocol & Protocol complexity

C.1 GGM Tree Analysis

Lemma 16. *For $n, \sigma \in \mathbb{N}$, the set $\mathcal{H}_\sigma^n := \{\mathbf{z} \mapsto \langle \mathbf{z}, \boldsymbol{\xi} \rangle \mid \xi \in \mathbb{F}_{2^\sigma}^n\}$ of functions $\mathbb{F}_{2^\sigma}^n \to \mathbb{F}_{2^\sigma}$ is a universal family of hash functions, i.e., for any two $\mathbf{x} \neq \mathbf{y} \in \mathbb{F}_{2^\sigma}^n$, we have $\Pr_{h \in_R \mathcal{H}_\sigma^n}[h(\mathbf{x}) = h(\mathbf{y})] = 2^{-\sigma}$.*

Proof. Halevi and Krawczyk [HK97] give the proof for Δ-universality for prime fields, but it works in the same way for normal universality and other finite fields: Let $\mathbf{x} \neq \mathbf{y} \in \mathbb{F}_{2^\sigma}^n$, and let $\mathbf{z}_h$ denote the vector corresponding to $h \in \mathcal{H}_\sigma^n$. W.l.o.g.

assume $x_1 \neq y_1$. Then

$$\Pr_{h\in_R\mathcal{H}^n_\sigma}[h(\mathbf{x}) = h(\mathbf{y})] = \Pr_{h\in_R\mathcal{H}^n_\sigma}[\langle \mathbf{z}_h, \mathbf{x}\rangle = \langle \mathbf{z}_h, \mathbf{y}\rangle]$$
$$= \Pr_{h\in_R\mathcal{H}^n_\sigma}\left[z_{h,1}\cdot(x_1-y_1) = -\sum_{i=2}^{n} z_{h,i}\cdot(x_i-y_i)\right] = 2^{-\sigma},$$

since $z_{h,1}\cdot(x_1-y_1)$ is uniform in $\mathbb{F}_{2^\sigma}$ and independent of the right-hand side. □

Lemma 17. *For $n, \sigma \in \mathbb{N}$, and any $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{F}_{2^\sigma}$, the collision probability is*

$$\Pr_{h\in_R\mathcal{H}^n_\sigma}\left[\exists i, j \in [n] \;.\; \mathbf{z}_i \neq \mathbf{z}_j \wedge h(\mathbf{z}_i) = h(\mathbf{z}_j)\right] \leq \frac{n(n-1)}{2}\cdot 2^{-\sigma} \leq 2^{-(\sigma - 2\log(n)+1)}.$$

Proof. We first apply the union bound and then use that $\mathcal{H}^n_\sigma$ is a universal family of hash functions:

$$\Pr_{h\in_R\mathcal{H}^n_\sigma}[\exists i, j \in [n] \;.\; \mathbf{z}_i \neq \mathbf{z}_j \wedge h(\mathbf{z}_i) = h(\mathbf{z}_j)]$$
$$= \Pr_{h\in_R\mathcal{H}^n_\sigma}\left[\bigvee_{i,j\in[n]} \mathbf{z}_i \neq \mathbf{z}_j \wedge h(\mathbf{z}_i) = h(\mathbf{z}_j)\right]$$
$$\leq \sum_{1\leq i<j\leq n} \Pr_{h\in_R\mathcal{H}^n_\sigma}[\mathbf{z}_i \neq \mathbf{z}_j \wedge h(\mathbf{z}_i) = h(\mathbf{z}_j)] = \frac{n(n-1)}{2}\cdot 2^{-\sigma}$$
$$\leq \frac{n^2}{2}\cdot 2^{-\sigma} = \frac{(2^{\log n})^2}{2}\cdot 2^{-\sigma} = 2^{2\log(n)-1}\cdot 2^{-\sigma} = 2^{-(\sigma-2\log(n)+1)} \quad \square$$

Theorem 18 (Theorem 5, restated). *Let $n = 2^h \in \mathbb{N}$, $\sigma' = \sigma + 2h$, and G, G' as above, and let $\mathcal{A}$ be any time adversary. If G' is right-half injective, then $\mathcal{A}$ can win the game in Definition 4 with probability at most $2^{-(\sigma+1)}$.*

Proof. If $\mathcal{A}$ wins, then there exists indices $\alpha \neq \alpha' \in I$ such that $\mathbf{v}^\alpha$ and $\mathbf{v}^{\alpha'}$ are not consistent. So we have an index $i \in [n] \setminus \{\alpha, \alpha'\}$ with $v_i^\alpha \neq v_i^{\alpha'}$. The values v_i^α and $v_i^{\alpha'}$ were derived from the keys $K_i^{h,\alpha}$ and $K_i^{h,\alpha'}$ using G':

$$\mathsf{G}'(K_i^{h,\alpha}) = (v_i^\alpha, c_i^\alpha) \qquad \mathsf{G}'(K_i^{h,\alpha'}) = (v_i^{\alpha'}, c_i^{\alpha'})$$

Since $v_i^\alpha \neq v_i^{\alpha'}$, we have $K_i^{h,\alpha} \neq K_i^{h,\alpha'}$. Due to the right-half injectivity of G' it follows $c_i^\alpha \neq c_i^{\alpha'}$ and, thus, also $\mathbf{t}^\alpha \neq \mathbf{t}^{\alpha'}$. Finally, $\langle \boldsymbol{\xi}, \mathbf{t}^\alpha\rangle = \langle \boldsymbol{\xi}, \mathbf{t}^{\alpha'}\rangle = \Gamma$ must hold.

By Lemma 16, the function $h(\mathbf{t}) := \langle \boldsymbol{\xi}, \mathbf{t}\rangle$ is a universal hash function sampled uniformly from the family $\{\mathbf{t} \mapsto \langle \boldsymbol{\xi}, \mathbf{t}\rangle \mid \boldsymbol{\xi} \in \mathbb{F}^n_{2^{\sigma'}}\}$. Note that Step 4 is deterministic and the $(\mathbf{v}^\alpha, \mathbf{t}^\alpha)$ depend only on the values that $\mathcal{A}$ produced in Step 1. This implies that $\boldsymbol{\xi}$ is independent of the $\mathbf{t}^\alpha$. So the adversary can only win the game if there is a collision among the $\mathbf{t}^\alpha$ under the randomly sampled hash function h. By Lemma 17, we can bound this probability by $2^{-(\sigma'-2h+1)} = 2^{-(\sigma+1)}$. □

C.2 Proof of Theorem 6

Proof of Theorem 6. First we cover the case of a corrupted sender, then that of a corrupted receiver.

Malicious Sender. The simulation is setup as follows: $\mathcal{S}$ simulates a party $\mathsf{P_S}^*$ in its head and gives control to $\mathcal{Z}$, and sends $(\mathsf{corrupt}, \mathsf{P_S})$ to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$. It also simulates instances of $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, $\mathcal{F}_{\mathsf{OT}}$, and $\mathcal{F}_{\mathsf{EQ}}$. Since the calls to (Init) and (Extend) are simply forwarded to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, $\mathcal{S}$ can just simulate the interaction with the ideal functionality. The main part of proof is the simulation of $(\mathsf{SP\text{-}Extend}, n)$. Let $h := \lceil \log n \rceil$.

1. $\mathcal{S}$ simulates the call $(\mathsf{Extend}, 1)$ to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ and receives $a, c \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_S}^*$.
2. Receive $a' \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_S}^*$. Compute $\beta := a' + a$ and set $\delta := c$.
3. Compute $k \leftarrow \mathsf{GGM.KeyGen}(1^\kappa)$ and execute $(\mathbf{v}, \mathbf{t}, (\overline{K}^i_0, \overline{K}^i_1)_{i\in[h]}, \overline{K}^{h+1}_1) \leftarrow \mathsf{GGM.Gen}(n, k)$.
4. Send $\overline{K}^{h+1}_1$ to $\mathsf{P_S}^*$.
5. Simulate invocation of $\mathcal{F}_{\mathsf{OT}}$: Record $\mathsf{P_S}^*$'s inputs $\overline{\alpha}_1, \ldots, \overline{\alpha}_h$, and send $\overline{K}^i_{\overline{\alpha}_i}$ for $i \in [h]$ to $\mathsf{P_S}^*$. Compute $\alpha := \sum_{i=0}^{h-1} 2^i \cdot \alpha_{h-i} \in [n]$.
6. Receive $\boldsymbol{\xi} \in \mathbb{F}^n_{2^{s'}}$ from $\mathsf{P_S}^*$ with $s' := \sigma + 2h$.
7. Compute $\Gamma := \langle \boldsymbol{\xi}, \mathbf{t} \rangle$ and send Γ to $\mathsf{P_S}^*$.
8. Define $\mathbf{u} \in \mathbb{Z}^n_{2^\ell}$ such that $u_\alpha = \beta$ and $u_i = 0$ for $i \in [n] \setminus \{\alpha\}$.
9. Sample $d \in_R \mathbb{Z}_{2^\ell}$ and send it to $\mathsf{P_S}^*$.
10. Define $\mathbf{w} \in \mathbb{Z}^n_{2^\ell}$ such that $w_i = v_i$ for $i \in [n] \setminus \{\alpha\}$ and $w_\alpha = \delta - d - \sum_{i\in[n]\setminus\{\alpha\}} w_i$.
11. $\mathcal{S}$ simulates the second call $(\mathsf{Extend}, 1)$ to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ and receives $x, z \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_S}^*$.
12. Receive $\boldsymbol{\chi} \in \{0,1\}^n$ from $\mathsf{P_S}^*$. If $\mathrm{HW}(\boldsymbol{\chi}) \neq \frac{n}{2}$, abort.
13. Receive $x^* \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_S}^*$, and compute $x' := x + x^*$ (for an honest sender, we have $x' = \chi_\alpha \cdot \beta$).
14. Compute $V_{\mathsf{P_S}} := \sum_{i\in[n]} \chi_i \cdot w_i - z \in \mathbb{Z}_{2^\ell}$ (the honest $\mathsf{P_S}$'s input to $\mathcal{F}_{\mathsf{EQ}}$), and record $\mathsf{P_S}^*$'s actual input $V'_{\mathsf{P_S}} = V_{\mathsf{P_S}} + \varepsilon$ to $\mathcal{F}_{\mathsf{EQ}}$.
15. If $x' = \chi_\alpha \cdot \beta$:
    - If $V'_{\mathsf{P_S}} = V_{\mathsf{P_S}}$: Simulate successful equality check, and send $\mathbf{u}, \mathbf{w}$ as $\mathsf{P_S}$'s outputs to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$.
    - If $V'_{\mathsf{P_S}} \neq V_{\mathsf{P_S}}$: Simulate failing equality check and abort.
16. If $x' = \chi_\alpha \cdot \beta + \eta$ with $\eta \neq 0 \pmod{2^\ell}$: Let $v \in \mathbb{N}$ be maximal such that $2^v \mid \eta$. It must be $v < r$.
    - If $2^v \nmid \varepsilon$, then simulate a failing equality check and abort.
    - If $2^v \mid \varepsilon$, then compute

$$\Delta' := \frac{\varepsilon}{2^v} \cdot \left(\frac{\eta}{2^v}\right)^{-1} \in \mathbb{Z}_{2^{r-v}}$$

      where the division is computed over $\mathbb{Z}$. If $r - v > s$ and $\Delta' \geq 2^s$, then simulate a failing equality check and abort. Otherwise set $s' := \min(s, r -$

v) and send (Guess, Δ', s') to $\mathcal{F}^{\ell,s}_{\text{sp-vole2k}}$. If it returns success, then simulate a successful equality check. If it aborts, then simulate a failing equality check and abort.

17. If $\mathsf{P_S}^*$ sends (Guess, $\widetilde{\Delta}$) to the simulated $\mathcal{F}^{\ell,s}_{\text{vole2k}}$, then $\mathcal{S}$ sends (Guess, $\widetilde{\Delta}$, s) to $\mathcal{F}^{\ell,s}_{\text{sp-vole2k}}$ and returns the answer to $\mathsf{P_S}^*$. If $\mathcal{F}^{\ell,s}_{\text{sp-vole2k}}$ aborts, then $\mathcal{S}$ also aborts.

Claim (Indistinguishability of the Simulation for Corrupted Sender). No PPT environment $\mathcal{Z}$ that chooses to corrupt the sender $\mathsf{P_S}$ can distinguish the real execution of the protocol from the simulation described above, except with probability $\mathsf{negl}(\kappa)$.

Proof of Claim. The simulation of (Init) and (Extend), i.e., sending (Init) and (Extend) to $\mathcal{F}^{\ell,s}_{\text{vole2k}}$, is identical to what would happen in the real protocol. Now we consider the simulation of (SP-Extend, n):

The call (Extend, 1) to the simulated $\mathcal{F}^{\ell,s}_{\text{vole2k}}$ results in $\mathsf{P_S}^*$ obtaining two values of its choice $a, c \in \mathbb{Z}_{2^\ell}$ – as in the real protocol. $\mathcal{S}$ creates the GGM tree honestly. Hence, the message $\overline{K}_1^{h+1}$ from the simulated receiver and the outputs that $\mathsf{P_S}^*$ receives from $\mathcal{F}_{\mathsf{OT}}$ are consistent and distributed as in the real protocol, and Γ is computed in the same way the real receiver would do it.

We leave the message d aside for a while, and focus on the remaining protocol first. From the second call (Extend, 1) to the simulated $\mathcal{F}^{\ell,s}_{\text{vole2k}}$, $\mathsf{P_S}^*$ again obtains two values of its choice $x, z \in \mathbb{Z}_{2^\ell}$.

Finally, we must make sure that the simulated $\mathcal{F}_{\mathsf{EQ}}$ behaves as in the real protocol:

The honest $\mathsf{P_S}$ would input $V_{\mathsf{P_S}} := \sum_{i\in[n]} \chi_i \cdot w_i - z \in \mathbb{Z}_{2^\ell}$ to $\mathcal{F}_{\mathsf{EQ}}$, whereas the corrupted sender can send some arbitrary $V'_{\mathsf{P_S}} = V_{\mathsf{P_S}} + \varepsilon$. The $\mathsf{P_R}$ normally computes its input as $V_{\mathsf{P_R}} := \sum_{i\in[n]} \chi_i \cdot v_i - y \in \mathbb{Z}_{2^\ell}$, where y was computed as $y := y^* - \Delta \cdot x^*$, Δ and y^* were received from $\mathcal{F}^{\ell,s}_{\text{vole2k}}$ and x^* from $\mathsf{P_S}^*$. So, in the simulation $\mathcal{S}$ does not know the right values of y^* and Δ.

$\mathcal{S}$ computes $x' := x + x^*$ (Step 13) and as noted above, we would have $x' = \chi_\alpha \cdot \beta$ if $\mathsf{P_S}^*$ behaved honestly. We obtain $\alpha \in [n]$ from the simulation of $\mathcal{F}_{\mathsf{OT}}$ (Step 5) and have received $\boldsymbol{\chi}$ from $\mathsf{P_S}^*$ (Step 12), so we know χ_α. $\mathcal{S}$ computes $\beta := a' + a$ from $\mathsf{P_S}^*$'s output a from $\mathcal{F}^{\ell,s}_{\text{vole2k}}$ and its message a' to $\mathsf{P_R}$ (Step 2). While an honest $\mathsf{P_S}$ would compute $a' := \beta - a$, sending an arbitrary a' is just equivalent to choosing a different value for β. Hence, $\mathcal{S}$ knows $\chi_\alpha \cdot \beta$ and can check whether $x' = \chi_\alpha \cdot \beta$ ($\star$) holds.

If the equality ($\star$) holds (Step 15), $\mathsf{P_R}$ would input $V_{\mathsf{P_R}} = V_{\mathsf{P_S}}$ to $\mathcal{F}_{\mathsf{EQ}}$. Hence, we can simulate a successful equality check if $V'_{\mathsf{P_S}} = V_{\mathsf{P_S}}$, and a failing equality check with an abort otherwise – as in the real protocol.

On the other hand, if the equality ($\star$) does not hold (Step 16), define $\eta \neq 0 \mod 2^\ell$ such that $x' = \chi_\alpha \cdot \beta + \eta$. This means, $\mathsf{P_S}^*$ has send a corrupted x^*, but the equality check might nevertheless pass if the errors ε in $V'_{\mathsf{P_S}}$ and η in x' cancel out.

On one hand, we have $\mathsf{P_R}$'s input $V_{\mathsf{P_R}}$ to $\mathcal{F}_{\mathsf{EQ}}$ which depends on η:

$$\begin{aligned} V_{\mathsf{P_R}} &= \sum_{i\in[n]} \chi_i \cdot v_i - y \\ &= \sum_{i\in[n]} \chi_i \cdot v_i - y^* + \Delta \cdot x^* \\ &= \sum_{i\in[n]} \chi_i \cdot v_i - y^* + \Delta \cdot (x' - x) \\ &= \sum_{i\in[n]} \chi_i \cdot v_i - y^* + \Delta \cdot (\chi_\alpha \cdot \beta + \eta - x) \\ &= V_{\mathsf{P_S}} + \Delta \cdot \eta \end{aligned}$$

On the other hand, we have $\mathsf{P_S}^*$'s input $V'_{\mathsf{P_S}} = V_{\mathsf{P_S}} + \varepsilon$. So the equality test should pass if and only if

$$\Delta \cdot \eta = \varepsilon \pmod{2^\ell} \tag{7}$$

holds. While $\mathcal{S}$ does not know the right value of Δ, it can use the global key query of $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$.

As above, define $v \in \mathbb{N}$ such that 2^v is the largest power of two that divides η. For Equation (7) to hold, 2^v must also divide ε. If this is not the case, $\mathcal{S}$ can safely simulate a failing equality test. Assuming 2^v divides both η and ε, we can divide both sides of Equation (7) by 2^v and reduce the modulus accordingly.

$$\Delta \cdot \frac{\eta}{2^v} = \frac{\varepsilon}{2^v} \pmod{2^{r-v}}$$

Since $\frac{\eta}{2^v}$ must be odd, we can solve for Δ.

$$\Delta = \frac{\varepsilon}{2^v} \cdot \left(\frac{\eta}{2^v}\right)^{-1} \pmod{2^{r-v}}$$

Let $\Delta' := \frac{\varepsilon}{2^v} \cdot \left(\frac{\eta}{2^v}\right)^{-1} \bmod 2^{r-v}$ be the result of the computation. We know that $\Delta \in \mathbb{Z}_{2^s}$. So, if $r - v \geq s$, then it must be $\Delta' = \Delta$ for the equality check to hold, and the abort in case $\Delta' \geq 2^s$ is safe. If $r - v < s$, then Δ' must consist of the lower $r - v$ bits of Δ for the check to hold. Hence, the global key query (Guess, Δ', $\min(s, r - v)$) of $\mathcal{S}$ results in an abort of $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ exactly when the equality test would fail.

To summarize, all messages (and abort) are distributed perfectly indistinguishable in protocol and simulation, except for d. We now show that any distinguisher $\mathcal{Z}$ for malicious senders can break the pseudorandomness (Definition 2) of the PPRF. For this, consider that in the real protocol $\mathsf{P_R}$ computes $d := \gamma - \sum_{j=1}^{n} v_j \in \mathbb{Z}_{2^\ell}$, where $\gamma := b - \Delta \cdot a'$ and $\mathsf{P_R}$ received Δ, b from $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ and a' from $\mathsf{P_S}$. On the other hand, in the simulation it is sampled uniformly at random as $d \in_R \mathbb{Z}_{2^\ell}$. The reason is that in the simulation, $\mathcal{S}$ never learns the secret key Δ that the ideal $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ has generated for the honest $\mathsf{P_R}$ – only the environment learns it.

We can now use any PPT environment $\mathcal{Z}$ that can distinguish both to break pseudorandomness as follows: First, observe that $\mathcal{Z}$ is fixed and that a reduction can make manipulations to the inner state of its security game. We construct a special simulator $\widehat{\mathcal{S}}$. Consider an execution of $\mathcal{Z}$ using $\widehat{\mathcal{S}}$ and let it first extract the point α where $\mathsf{P_S}$ will puncture the PPRF in the protocol from the hybrid $\mathcal{F}_{\mathsf{OT}}$.

$\widehat{\mathcal{S}}$ forwards α to the PPRF security game as in Definition 2 and obtains all seeds $\overline{K}^i$ for keys that it will input into the $\mathcal{F}_{\mathsf{OT}}$ to deliver to the sender.

It also obtains a point v^*_α which by the security game is either $F(k,\alpha)$ or uniformly random. Since our reduction can control the security experiment of $\mathcal{Z}$ and therefore controls $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ it has access to Δ that is generated by the functionality. It computes d as $d = \gamma - v^*_\alpha - \sum_{j=1,j\neq\alpha}^{n} v_j$ and uses d as its message to $\mathsf{P_S}$. For all other purposes, $\widehat{\mathcal{S}}$ just acts like $\mathcal{S}$.

By construction, if v^*_α is uniformly random then d is distributed as in the simulation, while if $v^*_\alpha = F(k,\alpha)$ then d is identical to the real protocol. Hence any $\mathcal{Z}$ that distinguishes both breaks Definition 2. By assumption, any PPT algorithm (in κ) can only do so with probability at most $\mathsf{negl}(\kappa)$. ■

Malicious Receiver. The simulation is setup as follows: $\mathcal{S}$ simulates a party $\mathsf{P_R}^*$ in its head and gives control to $\mathcal{Z}$, and sends $(\mathsf{corrupt}, \mathsf{P_R})$ to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$. It also simulates instances of $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, $\mathcal{F}_{\mathsf{OT}}$, and $\mathcal{F}_{\mathsf{EQ}}$.

For the call to (Init), $\mathcal{S}$ receives $\Delta \in \mathbb{Z}_{2^s}$ from $\mathsf{P_R}^*$, and sends Δ to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$. Calls to (Extend) are simply forwarded to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, and $\mathcal{S}$ can just simulate the interaction with the ideal functionality. The main part of proof is the simulation of $(\mathsf{SP\text{-}Extend}, n)$. Let $h := \lceil \log n \rceil$.

1. $\mathcal{S}$ simulates the call $(\mathsf{Extend}, 1)$ to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ and receives $b \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_R}^*$.
2. Send $a' \in_R \mathbb{Z}_{2^\ell}$ to $\mathsf{P_R}^*$.
3. Receive $\overline{K}_1^{h+1}$ from $\mathsf{P_R}^*$.
4. Simulates the calls to $\mathcal{F}_{\mathsf{OT}}$ and records $\mathsf{P_R}^*$'s inputs $(\overline{K}^i_0, \overline{K}^i_1)_{i\in[h]}$.
5. Sample $\boldsymbol{\xi} \in_R \mathbb{F}^n_{2^{s'}}$ with $s' := \sigma + 2h$, and send $\boldsymbol{\xi}$ to $\mathsf{P_R}^*$.
6. Receive Γ from $\mathsf{P_R}^*$.
7. For $\alpha \in [n]$, compute $\mathbf{v}^\alpha \leftarrow \mathsf{GGM.Eval}(n, \alpha, (\overline{K}^1_{\overline{\alpha}_1}, \ldots, \overline{K}^h_{\overline{\alpha}_h}, \overline{K}^{h+1}))$.
8. Define

$$I := \left\{\alpha \in [n] \mid \mathsf{GGM.Check}(n, \alpha, (\overline{K}^1_{\overline{\alpha}_1}, \ldots, \overline{K}^h_{\overline{\alpha}_h}, \overline{K}^{h+1}), \boldsymbol{\xi}, \Gamma) = \top\right\}.$$

   $\mathcal{S}$ sends I to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$. If it aborts, then simulate the abort of $\mathsf{P_S}$.
9. If the outputs $\mathbf{v}^\alpha$ are not consistent, i.e., $v^\alpha_j \neq v^{\alpha'}_j$ for some $\alpha \neq \alpha' \in I$ and $j \in [n] \setminus \{\alpha, \alpha'\}$, then abort the simulation (by Theorem 5, this should happen with negligible probability).
10. Case $|I| = 1$: $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ sends $(\mathsf{success}, \beta)$. With this information, we can basically follow the remaining protocol:

(a) Note that we now know the real α that $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$ has chosen since $I = \{\alpha\}$. Hence, we can compute $\mathbf{u} \in \mathbb{Z}_{2^\ell}^n$ such that $u_\alpha = \beta$ and $u_i = 0$ for all $i \neq \alpha$.
(b) Moreover, compute $a := \beta - a'$, and $\delta := \Delta \cdot a + b$.
(c) Receive $d' \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_R}^*$.
(d) Compute $\mathbf{w} \in \mathbb{Z}_{2^\ell}^n$ such that $w_i := v_i$ for all $i \neq \alpha$, and $w_\alpha := \delta - d' - \sum_{i \neq \alpha} w_i$.
(e) $\mathcal{S}$ simulates the second call $(\mathsf{Extend}, 1)$ to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ and receives $y^* \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_R}^*$.
(f) Sample $x \in_R \mathbb{Z}_{2^\ell}$ and set $z := \Delta \cdot x + y^*$.
(g) Sample $\boldsymbol{\chi} \in_R \{0,1\}^n$ with $\mathrm{HW}(\boldsymbol{\chi}) = \frac{n}{2}$ and compute $x^* := \chi_\alpha \cdot \beta - x$ and send them to $\mathsf{P_R}^*$.
(h) Compute $V_{\mathsf{P_S}} := \sum_{i=1}^n \chi_i \cdot w_i - z$. Record the value $V'_{\mathsf{P_R}}$ that $\mathsf{P_R}^*$ sends to $\mathcal{F}_{\mathsf{EQ}}$.
(i) If $V_{\mathsf{P_S}} \neq V'_{\mathsf{P_R}}$: Make $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$ abort by sending $(\mathsf{query}, \emptyset)$, and simulate failing equality test with $V_{\mathsf{P_S}}$ as $\mathsf{P_S}$'s input.
(j) If $V_{\mathsf{P_S}} = V'_{\mathsf{P_R}}$: Send $(\mathsf{continue})$ to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ and simulate successful equality test.
(k) Compute $\mathbf{v} := \mathbf{w} - \Delta \cdot \mathbf{u}$, and send $\mathbf{v}$ to $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$ as $\mathsf{P_R}$'s output.

11. Case $|I| > 1$: $\mathcal{F}_{\mathsf{sp\text{-}vole2k}}^{\ell,s}$ sends $(\mathsf{success})$
(a) Let $v_1, \ldots, v_n$ denote the leaves of the GGM tree. (We can recover these from $(v_j^\alpha)_{j \in [n] \setminus \{\alpha\}}$ and $(v_j^{\alpha'})_{j \in [n] \setminus \{\alpha'\}}$ for two different $\alpha, \alpha' \in I$.
(b) Define $d := (b - \Delta \cdot a') - \sum_{j=1}^n v_j \in \mathbb{Z}_{2^\ell}$, i.e., the value that an honest $\mathsf{P_R}$ would send. Receive $d' = d + \varepsilon \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_R}^*$, where ε denotes a possible error added by $\mathsf{P_R}^*$.
(c) $\mathcal{S}$ simulates the second call $(\mathsf{Extend}, 1)$ to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ and receives $y^* \in \mathbb{Z}_{2^\ell}$ from $\mathsf{P_R}^*$.
(d) Sample $\boldsymbol{\chi} \in_R \{0,1\}^n$ with $\mathrm{HW}(\boldsymbol{\chi}) = \frac{n}{2}$ and $x^* \in_R \mathbb{Z}_{2^\ell}$ and send them to $\mathsf{P_R}^*$.
(e) Compute $V_{\mathsf{P_R}} := \sum_{i=1}^n \chi_i \cdot v_i - (y^* - \Delta \cdot x^*) \in \mathbb{Z}_{2^\ell}$, i.e., the value an honest $\mathsf{P_R}$ would send to $\mathcal{F}_{\mathsf{EQ}}$. Record the actual value $V'_{\mathsf{P_R}}$ that $\mathsf{P_R}^*$ sends to $\mathcal{F}_{\mathsf{EQ}}$.
(f) If $\varepsilon = 0$:
  – If $V'_{\mathsf{P_R}} = V_{\mathsf{P_R}}$: Send $(\mathsf{continue})$ to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ and simulate successful equality test.
  – If $V'_{\mathsf{P_R}} \neq V_{\mathsf{P_R}}$: Send $(\mathsf{query}, \emptyset)$ to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$ and simulate failing equality test with $V_{\mathsf{P_R}}$ as $\mathsf{P_S}$'s input.
(g) If $\varepsilon \neq 0$:
  – If $V'_{\mathsf{P_R}} = V_{\mathsf{P_R}}$, define $J := \{i \in [n] \mid \chi_i = 0\}$ and $V^* := V_{\mathsf{P_R}} - \varepsilon$.
  – If $V'_{\mathsf{P_R}} = V_{\mathsf{P_R}} - \varepsilon$, define $J := \{i \in [n] \mid \chi_i = 1\}$ and $V^* := V_{\mathsf{P_R}}$.
  – If $V'_{\mathsf{P_R}} \notin \{V_{\mathsf{P_R}}, V_{\mathsf{P_R}} - \varepsilon\}$, define $J := \emptyset$. Sample $b \in_R \{0,1\}$ and set $V^* := V_{\mathsf{P_R}} - b \cdot \varepsilon$. Although the equality check will never pass, we cannot abort directly, since $\mathsf{P_R}^*$ learns $\mathsf{P_S}$'s input. Hence, we need to supply a value that looks right, where the error ε is subtracted with $1/2$ probability corresponding to the probability that $\chi_\alpha = 1$.

Send (query, J) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$. If it aborts, simulate failing equality test with V^* as $\mathsf{P_S}$'s input and simulate $\mathsf{P_S}$ aborting. Otherwise, receive $\alpha \in [n]$ from $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ and simulate successful equality test.

(h) We now know the correct value α chosen by the ideal functionality. Let $\mathbf{v}' \in \mathbb{Z}^n_{2^\ell}$ such that $v'_\alpha = v_\alpha - \varepsilon$ and $v'_i = v_i$ for $i \in [n] \setminus \{\alpha\}$. $\mathcal{S}$ sends $\mathbf{v}'$ as $\mathsf{P_R}$'s output to $\mathcal{F}^{2,s}_{\mathsf{vole2k}}$.

Claim (Indistinguishability of the Simulation for Corrupted Receiver). No PPT environment $\mathcal{Z}$ that chooses to corrupt the receiver $\mathsf{P_R}$ can distinguish the real execution of the protocol from the simulation described above except with probability $2^{-(\sigma+1)}$.

Proof of Claim. The simulation of Init, i.e., sending (Init) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, results again in a view of $\mathsf{P_R}^*$ which is identical to that in the real protocol. Note that $\mathcal{S}$ now learns the global key Δ since it is chosen by $\mathsf{P_R}^*$.

Now we consider the simulation of (Extend, n) with $n = 2^h$.

The call $(\mathsf{Extend}, 1)$ to the simulated $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ results in $\mathsf{P_R}^*$ obtaining a value of its choice $b \in \mathbb{Z}_{2^\ell}$ – as in the real protocol.

The first difference already occurs in Step 2, where the simulator sends a randomly samples $a' \in_R \mathbb{Z}_{2^\ell}$, whereas the real sender $\mathsf{P_S}$ would compute $a' := \beta - a$. In the simulation we don't know β, but a is distributed uniformly at random. Hence, from V^*'s view a' is distributed identically in both cases.

If we learn the correct β from $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ later on (we will, e.g., in Step 10), then we can compute $a := \beta - a'$ and $c := \Delta \cdot a + b$ and pretend that we received (a, c) from $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ during the first call to $(\mathsf{Extend}, 1)$, and these values will still be consistent with the view of $\mathsf{P_R}^*$.

The next Steps 3-6 simulate the transfer of the punctured PRF key. The message $\boldsymbol{\xi}$ send in Step 5 is sampled in the same way as the real $\mathsf{P_R}$ would do it.

Now we need to make sure that $\mathcal{S}$ simulates an abort if and only iff the consistency check of the GGM tree fails. In the simulation, we don't know the real value of α that the ideal functionality chooses, but we can collect all possible values of α for which the check passes in the set I. In the real execution, $\mathsf{P_S}$ would abort if and only if the real α is not contained in this set. Hence, in Step 8 we use the first query and send I to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$.

In the case that $\mathsf{P_R}^*$ has managed to create an inconsistent GGM tree which passes the check, then $\mathcal{S}$ aborts the simulation in Step 9, but by Theorem 5, this happens only with probability $2^{-(\sigma+1)}$. Therefore, we assume in the following that the GGM tree is consistent.

In case $|I| = 1$ (Step 10), we have recovered the correct value of α since it must be $I = \{\alpha\}$, and $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ sends use the correct value of β. So we can compute $\mathsf{P_S}$'s output $\mathbf{u}$ (Step 10a) and values a and δ which are consistent with the simulation (Step 10b). Then $\mathcal{S}$ can behaves in the same way as the real $\mathsf{P_S}$, and simulate the equality test accordingly. Since $\mathcal{S}$ can compute $\mathbf{w}$ in Step 10d and already knows Δ and $\mathbf{u}$, it can successfully recover $\mathbf{v}$ (Step 10k) and send it to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ to select $\mathsf{P_R}$'s output.

In case $|I| > 1$ (Step 11), we do not learn α and β from the query. We can, however, compute $\mathbf{v}$ in Step 11a, since we have (at least) two different $\mathbf{v}^{\alpha}$, $\mathbf{v}^{\alpha'}$. This allows us to compute the value d that an honest P_R would send, when we receive the (possibly maliciously chosen) message $d' = d+\varepsilon$ from $\mathsf{P}_\mathsf{R}{}^*$ (Step 11b). If $\varepsilon \neq 0$, then we know that $\mathsf{P}_\mathsf{R}{}^*$ is cheating, but we can only abort the simulation if $\mathsf{P}_\mathsf{R}{}^*$ is also caught by the check in the real protocol.

From the simulation of the $(\mathsf{Extend}, 1)$ call to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, we learn the value y^* that $\mathsf{P}_\mathsf{R}{}^*$ has chosen. The coefficient vector $\boldsymbol{\chi}$ is sampled uniformly at random as in the real protocol.

Since P_S uses the received d to compute the value of w at position α, the error ε is propagated and P_S instead computes $w'_\alpha = w_\alpha - \varepsilon$, and P_S's input to $\mathcal{F}_{\mathsf{EQ}}$ is $V'_{\mathsf{P}_\mathsf{S}} = V_{\mathsf{P}_\mathsf{S}} - \chi_\alpha \cdot \varepsilon$. Hence, to make the equality check pass, $\mathsf{P}_\mathsf{R}{}^*$ needs to adjust its input value $V'_{\mathsf{P}_\mathsf{R}}$ to account for the error ε iff $\chi_\alpha = 1$. Otherwise, it need to send the same value $V_{\mathsf{P}_\mathsf{R}}$ that the honest P_R would send. We can compute this value (Step 11e).

If $\varepsilon = 0$ ($\mathsf{P}_\mathsf{R}{}^*$ has sent the correctly computed d, covered in Step 11f), we know that the sender's input $V_{\mathsf{P}_\mathsf{S}}$ matches the honest P_R's input $V_{\mathsf{P}_\mathsf{R}}$. Hence, we can simulate a successful equality test if $\mathsf{P}_\mathsf{R}{}^*$'s actual input $V'_{\mathsf{P}_\mathsf{R}} = V_{\mathsf{P}_\mathsf{R}}$, and simulate an abort otherwise.

If $\varepsilon \neq 0$ ($\mathsf{P}_\mathsf{R}{}^*$ has sent an incorrect value $d' = d + \varepsilon$, covered in Step 11g), then we know that it must be $V_{\mathsf{P}_\mathsf{S}} = V_{\mathsf{P}_\mathsf{R}} - \chi_\alpha \cdot \varepsilon$ for the check to pass. So, if $V'_{\mathsf{P}_\mathsf{R}} \notin \{V_{\mathsf{P}_\mathsf{R}}, V_{\mathsf{P}_\mathsf{R}} - \varepsilon\}$, we can simulate a failing equality test, where the sender used either of the values as input with probability $1/2$. Since we do not know α, we cannot just lookup the value of χ_α in $\boldsymbol{\chi}$, but we can make a query to the ideal $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ whether $\alpha \in J$ for some index set J. If $V'_{\mathsf{P}_\mathsf{R}} = V_{\mathsf{P}_\mathsf{R}}$, we set J as the set of indices where $\boldsymbol{\chi}$ is 0, and if $V'_{\mathsf{P}_\mathsf{R}} = V_{\mathsf{P}_\mathsf{R}} - \varepsilon$, we set J as the set of indices where $\boldsymbol{\chi}$ is 1. Hence, J will contain exactly those values for α for which the the equality check would pass: if $V'_{\mathsf{P}_\mathsf{R}} = V_{\mathsf{P}_\mathsf{R}}$, then it must be $\chi_\alpha = 0$ so that the error ε disappears, and if $V'_{\mathsf{P}_\mathsf{R}} = V_{\mathsf{P}_\mathsf{R}} - \varepsilon$, then it must be $\chi_\alpha = 1$ so that ε is propagated. Hence, the equality test passes iff $\alpha \in J$. So we query the ideal $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ and simulate a failing test if it aborts. Otherwise, it sends us the value of α, and we can adjust the position α in $\mathsf{P}_\mathsf{R}{}^*$'s output $\mathbf{v}$ corresponding to the error ε (Step 11h).

The simulation is perfect unless $\mathcal{S}$ aborts it, which happens with probability at most $2^{-(\sigma+1)}$ if $\mathsf{P}_\mathsf{R}{}^*$ manages to create an inconsistent GGM tree which passes the check. ■

Theorem 6 follows from the combination of the two claims stating the indistinguishability of the simulations. □

C.3 Protocol Complexity

Let $n \in \mathbb{N}$, $h := \lceil \log n \rceil$, and $\sigma' := \sigma + 2h$. For one call to $(\mathsf{SP\text{-}Extend}, n)$, we

- use $2\times$ VOLE (of length $1 \times \mathbb{Z}_{2^\ell}$),
- use $h\times$ OT (on strings of length κ),

- use $1\times$ EQ for $\mathbb{Z}_{2^\ell}$ elements,
- transfer $3\times$ $\mathbb{Z}_{2^\ell}$ elements,
- transfer $2\times$ $\mathbb{F}_{2^{\sigma'}}$ elements, and
- transfer $2\times$ $\{0,1\}^\kappa$ PRG seeds.

Overall, using the equality test sketched in Section 2.3 and Silent OT [Boy+19; Yan+20; CRR21], we transfer $4\ell + 2\sigma + 4\lceil \log n \rceil + (5 + 2\lceil \log n \rceil)\kappa$ bit plus the costs of 2 VOLEs.

D Proof of Theorem 8

Proof of Theorem 8. We first consider the case of a corrupt sender, and then separately a corrupt receiver.

Malicious sender. We construct a simulator as follows. In the Init phase, the simulator first forwards (Init) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, and then receives $\mathbf{u}, \mathbf{w}$ from $\mathcal{A}$, as its input to the (Extend) command of $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$.

In the Extend phase, the simulator receives vectors $\mathbf{e}_i, \mathbf{c}_i$ from $\mathcal{A}$, for $i \in [t]$, and defines $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_t)$, $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_t)$. The simulator then computes $\mathbf{x}, \mathbf{z}$ as in the protocol, updates the vectors $\mathbf{u}, \mathbf{w}$ accordingly, and finally sends $(\mathbf{x}[m:n), \mathbf{z}[m:n)) \in \mathbb{Z}_{2^\ell}^\ell \times \mathbb{Z}_{2^\ell}^\ell$ as input to the $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ functionality.

Whenever $\mathcal{A}$ sends a key query command to $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$, the simulator sends the query to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ and forwards its response to $\mathcal{A}$. If $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ aborts, the simulator aborts.

Indistinguishability. Since there is no interaction in the protocol, and in the ideal world, the outputs of both parties are computed the exact same way as the real protocol, it is clear that the two executions are perfectly indistinguishable.

Malicious receiver. To simulate the Init phase, the simulator first receives Δ from $\mathcal{A}$ and forwards this to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, and then receives $\mathbf{v} \in \mathbb{Z}_{2^\ell}^m$ from $\mathcal{A}$.

In the Extend phase, the simulator samples a random noise vector $(\mathbf{e}_1, \ldots, \mathbf{e}_t)$, and uses this to respond to the leakage queries from $\mathcal{A}$, just as $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ would. If any query aborts, it sends abort to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$ and aborts. If the queries are successful, it receives $\mathbf{b}_i$ from $\mathcal{A}$, for $i \in [t]$, then defines $\mathbf{b} = (\mathbf{b}_1, \ldots, \mathbf{b}_t)$. It then computes $\mathbf{y} = \mathbf{v} \cdot \mathbf{A} + \mathbf{b}$, updates $\mathbf{v}$ as an honest P_R would, and sends the last $n - m$ entries of $\mathbf{y}$ to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$.

Indistinguishability. First, note that the probability of abort is identical in both the real and ideal executions, since the simulator responds to the leakage queries using a random noise vector, just as $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ does in the real execution. It remains to show that the distribution of the parties' outputs in the ideal execution is indistinguishable from the actual protocol.

For simplicity, we consider the case of a single call to Extend; handling multiple calls follows with a standard hybrid argument, and the fact that the LPN

secret used in subsequent calls is independent of previous outputs. Suppose there is an environment $\mathcal{Z}$ who controls an adversary $\mathcal{A}$ corrupting the receiver, and $\mathcal{Z}$ can distinguish between the two executions. We construct a distinguisher $\mathcal{D}$ for the leaky LPN assumption, as follows. $\mathcal{D}$ starts by simulating an execution of $\Pi^{\ell,s}_{\mathsf{vole2k}}$, as in the above simulation, until it reaches the leakage queries in the $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$ instances. Here, $\mathcal{D}$ receives t queries $I_1, \ldots, I_t \subset [n/t]$, forwards these to the leaky LPN challenger and uses its response to respond to $\mathcal{A}$. For the second set of leakage queries, it sends the sets $J_1, \ldots, J_t$ to the challenger, and again uses its response to simulate the response to $\mathcal{A}$. If the LPN challenger aborts, the simulation is aborted. Finally, $\mathcal{D}$ receives the vector $\mathbf{y}_b$ from the LPN challenger, and uses its last $n - m$ entries to define the honest sender's output $\mathbf{x}$, which is given to $\mathcal{Z}$; the output $\mathbf{z}$ is defined to be $\Delta \cdot \mathbf{x} + \mathbf{y}$.

Notice that the way the leakage queries are simulated is identical to the ideal functionality $\mathcal{F}^{\ell,s}_{\mathsf{sp\text{-}vole2k}}$. It follows that if $b = 0$ in the leaky LPN game, then the view of $\mathcal{Z}$ is identical to the real execution, while if $b = 1$, the view is the same as the ideal world. Therefore, the distinguishing advantage of $\mathcal{D}$ is the leaky LPN game is the same as that of $\mathcal{Z}$. □

E Solutions for Quadratic Equations Modulo 2^k

In this appendix, we provide some of the proofs for statements in Sections 5.1 & 5.2.

E.1 Proof of Proposition 9

Proof of Proposition 9. Clearly, if there exists one solution x then there are 3 more solutions $-x, x + 2^{\ell-1}$ and $-x + 2^{\ell-1}$. We now show that these are the only such solutions.

For the sake of contradiction, let y be such that $y^2 = a \bmod 2^\ell$. Then

$$x^2 - y^2 = (x - y)(x + y) = 0 \bmod 2^\ell.$$

Both x, y must be odd as a is odd, hence both $x + y$ and $x - y$ are a multiple of 2. If $x + y = 0$ or $x - y = 0$ then $x = \pm y$. Assuming this is false, then there exist odd numbers f, g as well as positive i, j such that

$$x + y = f2^i \ , \ x - y = g2^j \text{ and } i + j \geq \ell.$$

Since these equations hold over the integers, we additionally get that

$$(x + y) + (x - y) = 2x = f2^i + g2^j \Rightarrow x = f2^{i-1} + g2^{j-1}$$

where in the last step we divide over the integers. Since x must be odd, we have that either i or j must be 1. If $i = 1$ then

$$x - y = g2^{\ell-1} \Rightarrow x = y \bmod 2^{\ell-1}$$

whereas we get $x = -y \bmod 2^{\ell-1}$ if $j = 1$. □

E.2 Proof of Theorem 12

In the proof of Theorem 12, we will use the following statement.

Proposition 19. *Let $\ell, k, s \in \mathbb{N}^+$ so that $\ell \geq k+s$, and $\delta_0, \ldots, \delta_t \in \mathbb{Z}$ be values such that not all δ_i for $i > 0$ are $0 \bmod 2^k$. Then for any $j \in \{0, \ldots, \ell - k\}$*

$$\Pr\left[\delta_0 + \sum_{i\in[t]} \chi_i \cdot \delta_i = 0 \bmod 2^{k+j} \;\middle|\; \chi_1, \ldots, \chi_t \leftarrow \mathbb{Z}_{2^s}\right] \leq 2^{-\min(j,s)}.$$

The proof is an adaptation of the proof of Part *iii* of Lemma 1 of [Cra+18].

Proof of Proposition 19. Let $j \in \{0, \ldots, \ell - k\}$ be arbitrary. Without loss of generality assume that $\delta_t \neq 0 \pmod{2^k}$. Let $v \in \mathbb{N}$ be maximal such that $2^v \mid \delta_t$. This implies $v < k$. Define $S := \delta_0 + \sum_{i\in[t]} \chi_i \cdot \delta_i$ and $S' := -\delta_0 - \sum_{i\in[t-1]} \chi_i \cdot \delta_i$, and let $W := \min(\ell, e)$ where $e \in \mathbb{N}$ is maximal such that $2^e \mid S$.

Suppose $W = k + j$. By definition, $2^W \mid S$ which is equivalent to $S = 0 \pmod{2^{k+j}}$. Rewrite the equation as $\chi_t \cdot \delta_t = S' \pmod{2^{k+j}}$. By the definition of v, both sides must be multiples of 2^v. So we can divide by 2^v over the integers and reduce the modulus accordingly. Then $\delta/(2^v)$ is odd and, thus, invertible modulo 2^{k+j-v}. Since $v < k$, we have $k + j - v > j$, and can reduce the modulus further to 2^j:

$$\chi_t = \frac{S'}{2^v} \cdot \left(\frac{\delta_t}{2^v}\right)^{-1} \pmod{2^j}. \tag{8}$$

The left-hand side χ_t is distributed uniformly at random in $\mathbb{Z}_{2^s}$ and independent of the right-hand side. Hence, if $j < s$, then Equation (8) holds with probability at most 2^{-j}. For $j \geq s$, it holds with probability at most 2^{-s}. The proposition follows. □

Proof of Theorem 12. Let $a = \delta_0 + \sum_i \chi_i \delta_i \bmod 2^\ell$, then $\mathcal{A}$ wins iff $f(\Delta) = a\Delta^2 + b\Delta + c = 0 \bmod 2^\ell$. Let r be the largest value such that 2^r divides all a, b, c. We have that

$$\begin{aligned}
\Pr[\mathcal{A}\text{ wins}] \leq& \sum_{i=0}^{\ell} \Pr[\mathcal{A}\text{ wins}, r = i] \\
=& \Pr[\mathcal{A}\text{ wins} \mid r \in \{0, \ldots, k-1\}] \cdot \Pr[r \in \{0, \ldots, k-1\}] \\
&+ \sum_{i=k}^{\ell} \Pr[\mathcal{A}\text{ wins}, r = i] \\
\leq& \Pr[\mathcal{A}\text{ wins} \mid r \in \{0, \ldots, k-1\}] + \sum_{i=k}^{\ell} \Pr[\mathcal{A}\text{ wins}, r = i] \\
\leq& 2^{-\min\{(\ell-k)/2, s-1\}} + \sum_{j=0}^{\ell-k} \Pr[\mathcal{A}\text{ wins}, r = k + j]
\end{aligned} \tag{9}$$

Here, in the last step we use Lemma 11 where we set $\ell := \ell$ and $s' := \ell - k$.

By definition of r, if $r = k+j$ and $\mathcal{A}$ wins, then 2^{k+j} must divide a. Therefore

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ wins}, r = k + j] \\
=&\Pr[\mathcal{A} \text{ wins}, r = k + j, 2^{k+j} \text{ divides } a] \\
=&\Pr[\mathcal{A} \text{ wins}, r = k + j \mid 2^{k+j} \text{ divides } a] \cdot \Pr[2^{k+j} \text{ divides } a].
\end{aligned}$$

Claim. For $j \in \{0, \ldots, \ell-k\}$, the following inequalities holds (with $\lambda := \ell-k-s$):

a) $\Pr[2^{k+j} \text{ divides } a] \leq 2^{-\min\{j,s\}}$,
b) $\Pr[\mathcal{A} \text{ wins}, r = k + j \mid 2^{k+j} \text{ divides } a] \leq 2^{-\min\{(s+\lambda-j)/2,s\}+1}$,
c) $\Pr[\mathcal{A} \text{ wins}, r = k + j] \leq 2^{-s+1}$.

Proof of Claim.

a) Follows directly from Proposition 19.
b) By Lemma 11 for $\ell := \ell$, $r := k + j$, $s' := \ell - r - 1 = s + \lambda - j - 1$ and any $j \in \{0, \ldots, \ell - k\}$, there are at most $2^{\max\{(2s-s')/2,1\}} \leq 2^{\max\{(s+j-\lambda)/2,0\}+1}$ solutions in the range $\{0, \ldots, 2^s - 1\}$ to the equation $f(x) = 0 \pmod{2^\ell}$. Since there are 2^s choices for Δ, this means that

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}, r = k + j \mid 2^{k+j} \text{ divides } a] &\leq \frac{2^{\max\{(s+j-\lambda)/2,0\}+1}}{2^s} \\
&= 2^{\max\{(-s+j-\lambda)/2,-s\}+1} = 2^{-\min\{(s+\lambda-j)/2,s\}+1}.
\end{aligned}$$

c) Combining the previous two parts, we obtain

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ wins}, r = k + j] \\
=\ &\Pr[\mathcal{A} \text{ wins}, r = k + j \mid 2^{k+j} \text{ divides } a] \cdot \Pr[2^{k+j} \text{ divides } a] \\
\leq\ &2^{-\min\{(s+\lambda-j)/2,s\}+1} \cdot 2^{-\min\{j,s\}} \\
=\ &2^{-\min\{(s+\lambda-j)/2,s\}+1-\min\{j,s\}}.
\end{aligned}$$

For $j \in \{0, \ldots, s\}$, this can be simplified to

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}, r = k + j] &= 2^{-\min\{(s+\lambda-j)/2,s\}+1-j} \\
&= 2^{-\min\{(s+\lambda+j)/2,s+j\}+1} \\
&\overset{(\star)}{\leq} 2^{-\min\{(2s+j)/2,s+j\}+1} \\
&= 2^{-\min\{s+j/2,s+j\}+1} \\
&= 2^{-s-j/2+1} \\
&\leq 2^{-s+1},
\end{aligned}$$

where we use the fact that $\lambda \geq s$ at step $(\star)$, which follows from the assumption $\ell - k \geq 2s$ and the definition of λ. For $j \in \{s, \ldots, \ell - k\}$, we obtain the bound directly from Part a):

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ wins}, r = k + j] \\
&= \Pr[\mathcal{A} \text{ wins}, r = k + j \mid 2^{k+j} \text{ divides } a] \cdot \Pr[2^{k+j} \text{ divides } a] \\
&\leq \Pr[\mathcal{A} \text{ wins}, r = k + j \mid 2^{k+j} \text{ divides } a] \cdot 2^{-\min\{j,s\}} \\
&\leq 2^{-\min\{j,s\}} = 2^{-s} \leq 2^{-s+1}.
\end{aligned}$$

■

Continuing from Equation (9), we use the Claim proved above and the fact that $\ell \geq k + 2s$ to get

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}] &\overset{(9)}{\leq} 2^{-\min\{(\ell-k)/2, s-1\}} + \sum_{j=0}^{\ell-k} \Pr[\mathcal{A} \text{ wins}, r = k + j] \\
&\leq 2^{-\min\{s, s-1\}} + \sum_{j=0}^{\ell-k} 2^{-s+1} \\
&\leq 2^{-s+1} + \sum_{j=0}^{\ell-k} 2^{-s+1} \\
&\leq (\ell - k + 2) \cdot 2^{-s+1}.
\end{aligned}$$

□

E.3 Proof of Corollary 13

Proof of Corollary 13. Plugging in the values of s and ℓ into the winning probability stated in the theorem gives us:

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}] &\leq (\ell - k + 2) \cdot 2^{-s+1} \\
&= (k + 2s - k + 2) \cdot 2^{-s+1} \\
&= (2s + 2) \cdot 2^{-s+1} \\
&= (2\sigma + 2\log\sigma + 8) \cdot 2^{-\sigma - \log\sigma - 2}.
\end{aligned}$$

By taking the logarithm of both sides, we get

$$\begin{aligned}
\log \Pr[\mathcal{A} \text{ wins}] &\leq \log(2\sigma + 2\log(\sigma) + 8) - \sigma - \log(\sigma) - 2 \\
&\overset{(\star)}{\leq} \log(4\sigma) - \sigma - \log(\sigma) - 2 \\
&= \log(\sigma) + 2 - \sigma - \log(\sigma) - 2 \\
&= -\sigma,
\end{aligned}$$

where we use that $2\sigma + 2\log(\sigma) + 8 \leq 4\sigma$ $(\star)$ holds for all $\sigma \geq 7$. Hence, we have bounded the winning probability as $\Pr[\mathcal{A} \text{ wins}] \leq 2^{-\sigma}$. □

F Proof of Theorem 14

Proof of Theorem 14. We divide the proof of security into two parts. First we cover the case of a corrupted prover, then that of a corrupted verifier. In each case we define a PPT simulator $\mathcal{S}$.

Corrupted Prover: The simulation is set up as follows: $\mathcal{S}$ simulates a party $\mathcal{P}^*$ in its head and gives control to $\mathcal{Z}$, and sends (corrupt, $\mathcal{P}$) to $\mathcal{F}_{\mathsf{ZK}}^k$. It also simulates an instance of $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$. We assume that the circuit $\mathcal{C}$ is known.

1. Simulation of the preprocessing phase: $\mathcal{S}$ simulates the (Init) and (Extend, $n+t+2$) calls to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$. For (Init), it samples $\Delta \in_R \mathbb{Z}_{2^s}$. Since $\mathcal{P}^*$ acts as the sender $\mathsf{P_S}$ towards $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$, it is allowed to choose the sender's output of the (Extend) call. Hence, $\mathcal{S}$ receives $(\widetilde{\mu}_i, M[\mu_i]), (\widetilde{\nu}_j, M[\nu_j]), (\widetilde{o}, M[o]), (\widetilde{\pi}, M[\pi]) \in \mathbb{Z}_{2^\ell} \times \mathbb{Z}_{2^\ell}$ for $i \in [n]$ and $j \in [t]$ from $\mathcal{P}^*$. Then $\mathcal{S}$ can compute matching values $K[\mu_i], K[\nu_j], K[o], K[\pi] \in \mathbb{Z}_{2^\ell}$ according to Equation (4).
2. To simulate the online phase, $\mathcal{S}$ executes the steps of $\mathcal{V}$ while also keeping track of $\mathcal{P}^*$'s wire values:
   - For every circuit input $i \in [n]$ it receives $\delta_i \in \mathbb{Z}_{2^\ell}$ from $\mathcal{P}^*$ and computes $\mathcal{V}$'s part of $[w_i] := [\mu_i] + \delta_i$, as well as $\widetilde{w}_i := \widetilde{\mu}_i + \delta_i \in \mathbb{Z}_{2^\ell}$.
   - For every addition gate (α, β, γ) it computes $\mathcal{V}$'s part of $[w_\gamma] := [w_\alpha] + [w_\beta]$, as well as $\widetilde{w}_\gamma := \widetilde{w}_\alpha + \widetilde{w}_\beta \in \mathbb{Z}_{2^\ell}$ and $M[w_\gamma] := M[w_\alpha] + M[w_\beta]$.
   - For the ith multiplication gate (α, β, γ) it receives $d_i \in \mathbb{Z}_{2^\ell}$ from $\mathcal{P}^*$ and computes $\mathcal{V}$'s part of $[w_\gamma] := [\nu_i] + d_i$, as well as $\widetilde{w}_\gamma := \widetilde{\nu}_i + d_i \in \mathbb{Z}_{2^\ell}$. Additionally it computes $B_i := K[w_\alpha] \cdot K[w_\beta] + \Delta \cdot K[w_\gamma]$.
   - $\mathcal{S}$ sends $\boldsymbol{\chi} \in_R \mathbb{Z}_{2^s}^t$ to $\mathcal{P}^*$ and receives two values $U', V' \in \mathbb{Z}_{2^\ell}$ as response. It computes $W := \sum_{i \in [t]} \chi_i \cdot B_i + B^* \in \mathbb{Z}_{2^\ell}$ where $B^* := K[o]$.
   - If $W \neq U' + V' \cdot \Delta \pmod{2^\ell}$, then $\mathcal{S}$ sends (Prove, $\mathcal{C}$, $\bot$) on behalf of $\mathcal{P}^*$ to $\mathcal{F}_{\mathsf{ZK}}^k$ and simulates an aborting $\mathcal{V}$.
   - For the single output wire w_h, $\mathcal{S}$ already holds $K[w_h]$ and computes $K[z] := K[w_h] + 2^k \cdot K[\pi]$. Then it receives two values $\widetilde{z}, M[z] \in \mathbb{Z}_{2^\ell}$ from $\mathcal{P}^*$ and checks if $K[z] \stackrel{?}{=} M[z] + \widetilde{z} \cdot \Delta$ holds and $\widetilde{z} = 1 \pmod{2^k}$. If this is the case, then $\mathcal{S}$ sends (Prove, $\mathcal{C}$, $\mathbf{w}$) with $\mathbf{w} := (\widetilde{w}_i \bmod 2^k)_{i \in [n]}$ on behalf of $\mathcal{P}$ to $\mathcal{F}_{\mathsf{ZK}}^k$.

Since $\mathcal{S}$ behaves like an honest $\mathcal{V}$ towards $\mathcal{P}^*$, the view of $\mathcal{P}$ in the simulation perfectly matches its view in the real protocol. Now we need to show that $\mathcal{V}$ outputs the same in the simulation and in the real execution, except with negligible probability. If the honest $\mathcal{V}$ rejects the proof in the protocol, then also $\mathcal{V}$'s output of the ideal $\mathcal{F}_{\mathsf{ZK}}^k$ is false. Now we need to bound the probability that the honest $\mathcal{V}$ accepts the proof, but the ideal $\mathcal{F}_{\mathsf{ZK}}^k$ still outputs false, i.e., $\mathcal{P}^*$ has successfully fooled $\mathcal{V}$ into accepting a proof of a wrong statement.

A corrupted $\mathcal{P}^*$ not knowing a valid witness for the circuit can try to cheat in two ways: It can try to circumvent the multiplication check in Step 4 or it can try to open $[z]$ in Step 5 to an invalid value. The latter succeeds with probability at most 2^{-s} [Bau+21a].

Now we consider the former case: For the ith multiplication gate (α, β, γ), let $\widetilde{w}_\gamma = \widetilde{w}_\alpha \cdot \widetilde{w}_\beta + e_i \pmod{2^\ell}$, where $\widetilde{w}_\alpha, \widetilde{w}_\beta, \widetilde{w}_\gamma \in \mathbb{Z}_{2^\ell}$ are the values contained in the commitments $[w_\alpha], [w_\beta], [w_\gamma]$ and $e_i \in \mathbb{Z}_{2^\ell}$ is a possible error. Suppose that not all $e_i = 0 \pmod{2^k}$ for $i \in [t]$, i.e., there is an error in the lower k bits that we care about.

Then we have (all over $\mathbb{Z}_{2^\ell}$)

$$\begin{aligned} K[w_\gamma] &= M[w_\gamma] + \widetilde{w}_\gamma \cdot \Delta \\ &= M[w_\gamma] + (\widetilde{w}_\alpha \cdot \widetilde{w}_\beta + e_i) \cdot \Delta \\ &= M[w_\gamma] + (\widetilde{w}_\alpha \cdot \widetilde{w}_\beta) \cdot \Delta + e_i \cdot \Delta, \end{aligned}$$

and

$$\begin{aligned} B_i &= K[w_\alpha] \cdot K[w_\beta] - \Delta \cdot K[w_\gamma] \\ &= (M[w_\alpha] + \widetilde{w}_\alpha \cdot \Delta) \cdot (M[w_\beta] + \widetilde{w}_\beta \cdot \Delta) - \Delta \cdot (M[w_\gamma] + \widetilde{w}_\gamma \cdot \Delta) \\ &= (M[w_\alpha] + \widetilde{w}_\alpha \cdot \Delta) \cdot (M[w_\beta] + \widetilde{w}_\beta \cdot \Delta) - \Delta \cdot (M[w_\gamma] + (\widetilde{w}_\alpha \cdot \widetilde{w}_\beta + e_i) \cdot \Delta) \\ &= (M[w_\alpha] \cdot M[w_\beta]) + (\widetilde{w}_\alpha \cdot M[w_\beta] + M[w_\alpha] \cdot \widetilde{w}_\beta - M[w_\gamma]) \cdot \Delta - e_i \cdot \Delta^2 \\ &= A_{i,0} + A_{i,1} \cdot \Delta - e_i \cdot \Delta^2, \end{aligned}$$

where $A_{i,0}$ and $A_{i,1}$ denote the values that an honest $\mathcal{P}$ would compute. So now B_i is the result of evaluating a quadratic polynomial at Δ instead of a linear one. The equations for all gates are aggregated using the random linear combination:

$$\begin{aligned} W &= \sum_{i \in [t]} \chi_i \cdot B_i + B^* \\ &= \sum_{i \in [t]} \chi_i \cdot \left(A_{i,0} + A_{i,1} \cdot \Delta - e_i \cdot \Delta^2\right) + A_0^* + A_1^* \cdot \Delta \\ &= \left(\sum_{i \in [t]} \chi_i \cdot A_{i,0} + A_0^*\right) + \left(\sum_{i \in [t]} \chi_i \cdot A_{i,1} + A_1^*\right) \cdot \Delta - \left(\sum_{i \in [t]} \chi_i \cdot e_i\right) \cdot \Delta^2 \\ &= U + V \cdot \Delta - \left(\sum_{i \in [t]} \chi_i \cdot e_i\right) \cdot \Delta^2 \end{aligned} \tag{10}$$

Here, U, V denote the values that an honest $\mathcal{P}$ would send to $\mathcal{V}$. The corrupted $\mathcal{P}^*$ may choose to send some values $U' := U + e_U$ and $V' := V + e_V$ instead. Note that $\mathcal{V}$ accepts the proof if $W = U' + V' \cdot \Delta$ holds. By subtracting Equation (10) from this, we get that $\mathcal{V}$ accepts if

$$0 = e_U + e_V \cdot \Delta + \left(\sum_{i \in [t]} \chi_i \cdot e_i\right) \cdot \Delta^2 \pmod{2^\ell}. \tag{11}$$

holds.

The steps in the protocol corresponding exactly to the game defined in Theorem 12: Initially, $\Delta \in_R \mathbb{Z}_{2^s}$ is sampled. When committing to the results of the multiplications, $\mathcal{P}^*$ defines the error values $e_1, \ldots, e_t \in \mathbb{Z}_{2^\ell}$ where not all of the e_i are 0 modulo 2^k if $\mathcal{P}^*$ tries to cheat. After $\mathcal{P}^*$ has committed itself on the e_i, $\mathcal{V}$ samples the coefficients $\chi_1, \ldots, \chi_t \in_R \mathbb{Z}_{2^s}$ of the random linear combination. Finally, the prover responds by choosing values $e_U, e_V \in \mathbb{Z}_{2^\ell}$, and wins the game, i.e., cheats successfully, if Equation (11) holds. Hence, we can apply Corollary 13 to bound the probability that this happens with $2^{-\sigma}$.

By the union bound, no adversary can break the soundness of the protocol except with probability at most $2^{-s} + 2^{-\sigma} \leq 2^{-\sigma+1}$.

Corrupted Verifier: The simulation of the verifier's view is straightforward: $\mathcal{S}$ simulates a party $\mathcal{V}^*$ in its head and gives control to $\mathcal{Z}$, and sends (corrupt, $\mathcal{V}$) to $\mathcal{F}_{\mathsf{ZK}}^k$. It also simulates an instance of $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$. We assume that the circuit $\mathcal{C}$ to prove is known.

1. Simulation of the preprocessing phase: $\mathcal{S}$ simulates the (Init) and (Extend, $n+t+2$) calls to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$. Since $\mathcal{V}^*$ acts as the sender $\mathsf{P_S}$ towards $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$, it is allowed to choose its outputs. For (Init), $\mathcal{S}$ receives $\Delta \in \mathbb{Z}_{2^s}$ from $\mathcal{V}^*$, and $K[\mu_i], K[\nu_j], K[o], K[\pi] \in \mathbb{Z}_{2^\ell}$ for $i \in [n]$ and $j \in [t]$ from $\mathcal{V}^*$.
2. To simulate the online phase, $\mathcal{S}$ proceeds as follows:
   - For every circuit input $i \in [n]$ it sends a random $\delta_i \in_R \mathbb{Z}_{2^\ell}$ to $\mathcal{V}^*$, and computes $K[w_i] := K[\mu_i] - \delta_i \cdot \Delta$.
   - For the ith multiplication gate (α, β, γ) it sends a random $d_i \in_R \mathbb{Z}_{2^\ell}$ to $\mathcal{V}^*$, and computes $K[w_\gamma] := K[\nu_i] - d_i \cdot \Delta$ and $B_i := K[w_\alpha] \cdot K[w_\beta] + \Delta \cdot K[w_\gamma] \in \mathbb{Z}_{2^\ell}$ as the honest $\mathcal{V}$ would do.
   - $\mathcal{S}$ receives $\boldsymbol{\chi} \in \mathbb{Z}_{2^s}^t$ from $\mathcal{V}^*$.
   - It computes $W := \sum_{i \in [t]} \chi_i \cdot B_i + B^* \in \mathbb{Z}_{2^\ell}$ where $B^* = K[o]$. Then it samples $V' \in_R \mathbb{Z}_{2^\ell}$ and sets $U' := W - V' \cdot \Delta$, and sends (U', V') to $\mathcal{V}^*$.
   - For the output wire w_h, $\mathcal{S}$ already holds $K[w_h]$. It samples $\widetilde{\pi} \in_R \mathbb{Z}_{2^\ell}$, and computes $\widetilde{z} := 1 + 2^k \cdot \widetilde{\pi} \in \mathbb{Z}_{2^\ell}$ and $K[z] := K[w_h] + 2^k \cdot K[\pi] \in \mathbb{Z}_{2^\ell}$. Finally it sends $\widetilde{z}$ as well as $M[z] := K[z] - \Delta \cdot \widetilde{z} \in \mathbb{Z}_{2^\ell}$ to $\mathcal{V}^*$.

The view of $\mathcal{V}^*$ is distributed exactly as in the real execution of the protocol: The values δ_i and d_i are distributed uniformly in $\mathbb{Z}_{2^\ell}$ and therefore completely mask the circuit inputs and the output values of the multiplication gates, respectively. Moreover, the values U, V computed by the honest $\mathcal{P}$ are also distributed uniformly at random due to the masking with A_0^* and A_1^*, respectively, under the condition that $W = U + V \cdot \Delta$ holds. Hence, the values U', V' sent to $\mathcal{V}^*$ by $\mathcal{S}$ are distributed identically, since we can recover the value W' that an honest $\mathcal{V}$ would compute. Finally, the last message is a valid opening of a commitment [1] where the upper $\ell - k$ bits have been randomized using $[\pi]$. □

Zero-Knowledge Functionality $\mathcal{F}^k_{\mathsf{ZK\text{-}2}}$

Prove: On input (Prove, $\{f_1, \ldots, f_t\}$, $\mathbf{w}$) from $\mathcal{P}$ and (Verify, $\{f_1, \ldots, f_t\}$) from $\mathcal{V}$ where $f_1, \ldots, f_n$ are polynomials of degree ≤ 2 in n variables over $\mathbb{Z}_{2^k}$ and $\mathbf{w} \in \mathbb{Z}^n_{2^k}$: Send true to $\mathcal{V}$ iff $f_i(\mathbf{w}) = 0$ for all $i \in [t]$, and false otherwise.

Fig. 8. Ideal functionality for zero-knowledge proofs for sets of degree-2 polynomials.

QuarkSilver for general degree-2 relations $\Pi^k_{\mathsf{QS\text{-}2}}$

The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ have agreed on a set of polynomials in n variabled $f_1, \ldots, f_t \in \mathbb{Z}_{2^k}[X_1, \ldots, X_n]^{\leq 2}$, and $\mathcal{P}$ holds a witness $\mathbf{w} \in \mathbb{Z}^n_{2^k}$ so that $f_i(\mathbf{w}) = 0$ for all $i \in [t]$. Write $f_i = f_{i,0} + f_{i,1} + f_{i,2}$ where $f_{i,h}$ contains the degree-h terms of f_i.

Preprocessing phase The preprocessing phase is independent of $\mathcal{C}$ and just needs upper bounds on the number of inputs and multiplication gates of $\mathcal{C}$ as input.

1. $\mathcal{P}$ and $\mathcal{V}$ send (Init) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, and $\mathcal{V}$ receives $\Delta \in \mathbb{Z}_{2^s}$.
2. $\mathcal{P}$ and $\mathcal{V}$ send (Extend, n + 1) to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, which returns authenticated values $([\mu_i])_{i\in[n]}$ and $[o]$, where all $\widetilde{\mu}_i, \widetilde{o}, \in_R \mathbb{Z}_{2^\ell}$.

Online phase

1. $\mathcal{P}$ commits to its witness $\mathbf{w}$ by sending $\delta_i := w_i - \widetilde{\mu}_i$ for $i \in [n]$ to $\mathcal{V}$, and both parties locally compute $[w_i] := [\mu_i] + \delta_i$.
2. For each f_i, $i \in [t]$:
   - $\mathcal{V}$ computes $B_i := \sum_{h\in[0,2]} f_{i,h}(K[w_1], \ldots, K[w_n]) \cdot \Delta^{2-h}$
   - $\mathcal{P}$ defines $g_i(X) \in \mathbb{Z}_{2^\ell}[X]$ as $g_i(X) := \sum_{h\in[0,2]} f_{i,h}(M[w_1] + \widetilde{w}_1 \cdot X, \ldots, M[w_n] + \widetilde{w}_n \cdot X) \cdot X^{2-h}$ and computes coefficients $A_{i,h} \in \mathbb{Z}_{2^\ell}$ such that $g_i(X) = A_{i,0} + A_{i,1} \cdot X + A_{i,2} \cdot X^2$. Note that $A_{i,2} = f_i(w_1, \ldots, w_n) = 0$ if $\mathcal{P}$ is honest.
3. $\mathcal{P}$ and $\mathcal{V}$ run the following check:
   (a) Set $A^*_0 := M[o]$, $A^*_1 := \widetilde{o}$, and $B^* := K[o]$ so that $B^* = A^*_0 + A^*_1 \cdot \Delta$.
   (b) $\mathcal{V}$ samples $\boldsymbol{\chi} \in_R \mathbb{Z}^t_{2^s}$ and sends it to $\mathcal{P}$.
   (c) $\mathcal{P}$ computes $U := \sum_{i\in[t]} \chi_i \cdot A_{0,i} + A^*_0 \in \mathbb{Z}_{2^\ell}$ and $V := \sum_{i\in[t]} \chi_i \cdot A_{1,i} + A^*_1 \in \mathbb{Z}_{2^\ell}$, and sends (U, V) to $\mathcal{V}$.
   (d) $\mathcal{V}$ computes $W := \sum_{i\in[t]} \chi_i \cdot B_i + B^* \in \mathbb{Z}_{2^\ell}$, and accepts iff $W = U + V \cdot \Delta \pmod{2^\ell}$ holds.

Fig. 9. Zero-Knowledge protocol for circuit satisfiability in the $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$-hybrid model with $s := \sigma + \log(\sigma) + 3$ and $\ell := k + 2s$ for statistical security parameter σ.

G QuarkSilver for General Degree-2 Relations

We formally specify the ideal zero-knowledge functionality for degree-2 relations in Figure 8, and prove in the following that protocol $\Pi^k_{\mathsf{QS\text{-}2}}$ (Figure 9) realizes this functionality.

Theorem 20. *The protocol $\Pi^k_{\mathsf{QS\text{-}2}}$ (Figure 9) securely realizes the functionality $\mathcal{F}^k_{\mathsf{ZK\text{-}2}}$ in the $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$-hybrid model when instantiated with the parameters $s := \sigma + \log(\sigma) + 3$ and $\ell := k + 2s$: No unbounded environment $\mathcal{Z}$ can distinguish the real execution of the protocol from a simulated one except with probability $2^{-\sigma+1}$.*

Proof. The proof is similar to the proof of Theorem 14 in Appendix F. We divide the proof of security into two parts. First we cover the case of a corrupted prover, then that of a corrupted verifier. In each case we define a PPT simulator $\mathcal{S}$.

Corrupted Prover: The simulation is set up as follows: $\mathcal{S}$ simulates a party $\mathcal{P}^*$ in its head and gives control to $\mathcal{Z}$, and sends ($\mathsf{corrupt}$, $\mathcal{P}$) to $\mathcal{F}^k_{\mathsf{ZK\text{-}2}}$. It also simulates an instance of $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$. We assume that the circuit $\mathcal{C}$ is known.

1. Simulation of the preprocessing phase: $\mathcal{S}$ simulates the (Init) and (Extend, $n+1$) calls to $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$. For ($\mathsf{Init}$), it samples $\Delta \in_R \mathbb{Z}_{2^s}$. Since $\mathcal{P}^*$ acts as the sender $\mathsf{P_S}$ towards $\mathcal{F}^{\ell,s}_{\mathsf{vole2k}}$, it is allowed to choose the sender's output of the (Extend) call. Hence, $\mathcal{S}$ receives $(\widetilde{\mu}_i, M[\mu_i]), (\widetilde{o}, M[o]) \in \mathbb{Z}_{2^\ell} \times \mathbb{Z}_{2^\ell}$ for $i \in [n]$ from $\mathcal{P}^*$. Then $\mathcal{S}$ can compute matching values $K[\mu_i], K[o] \in \mathbb{Z}_{2^\ell}$ according to Equation (4).
2. To simulate the online phase, $\mathcal{S}$ executes the steps of $\mathcal{V}$ while also keeping track of $\mathcal{P}^*$'s values:
   - For every input $i \in [n]$ it receives $\delta_i \in \mathbb{Z}_{2^\ell}$ from $\mathcal{P}^*$ and computes $\mathcal{V}$'s part of $[w_i] := [\mu_i] + \delta_i$, as well as $\widetilde{w}_i := \widetilde{\mu}_i + \delta_i \in \mathbb{Z}_{2^\ell}$ and $M[w_i] := M[\mu_i]$.
   - For the ith polynomial f_i it computes B_i, $A_{i,0}$, and $A_{i,1}$ as in Step 2.
   - $\mathcal{S}$ sends $\boldsymbol{\chi} \in_R \mathbb{Z}^t_{2^s}$ to $\mathcal{P}^*$ and receives two values $U', V' \in \mathbb{Z}_{2^\ell}$ as response. It computes $W := \sum_{i\in[t]} \chi_i \cdot B_i + B^* \in \mathbb{Z}_{2^\ell}$ where $B^* := K[o]$.
   - If $W \neq U' + V' \cdot \Delta \pmod{2^\ell}$, then $\mathcal{S}$ sends (Prove, $\{f_1, \ldots, f_t\}$, $\perp$) on behalf of $\mathcal{P}^*$ to $\mathcal{F}^k_{\mathsf{ZK\text{-}2}}$ and simulates an aborting $\mathcal{V}$.
   - Otherwise $\mathcal{S}$ sends (Prove, $\{f_1, \ldots, f_t\}$, $\mathbf{w}$) with $\mathbf{w} := (\widetilde{w}_i \bmod 2^k)_{i\in[n]}$ on behalf of $\mathcal{P}$ to $\mathcal{F}^k_{\mathsf{ZK\text{-}2}}$.

Since $\mathcal{S}$ behaves like an honest $\mathcal{V}$ towards $\mathcal{P}^*$, the view of $\mathcal{P}$ in the simulation perfectly matches its view in the real protocol. Now we need to show that $\mathcal{V}$ outputs the same in the simulation and in the real execution, except with negligible probability. If the honest $\mathcal{V}$ rejects the proof in the protocol, then also $\mathcal{V}$'s output of the ideal $\mathcal{F}^k_{\mathsf{ZK\text{-}2}}$ is false. Now we need to bound the probability that the honest $\mathcal{V}$ accepts the proof, but the ideal $\mathcal{F}^k_{\mathsf{ZK}}$ still outputs false, i.e., $\mathcal{P}^*$ has successfully fooled $\mathcal{V}$ into accepting a proof of a wrong statement.

A corrupted $\mathcal{P}^*$ not knowing a valid witness for the circuit can try to circumvent the check in Step 3: For the ith polynomial f_i, let $0 = f(\widetilde{w}_1, \ldots, \widetilde{w}_n) \pmod{2^\ell} + e_i$, where $\widetilde{w}_j \in \mathbb{Z}_{2^\ell}$ is the value contained in the commitment $[w_j]$, for $j \in [n]$, and $e_i \in \mathbb{Z}_{2^\ell}$ is a possible error. Suppose that not all $e_i = 0 \pmod{2^k}$ for $i \in [t]$, i.e., there is an error in the lower k bits that we care about.

Then we have (over $\mathbb{Z}_{2^\ell}$)

$$\begin{aligned} B_i &= \sum_{h\in[0,2]} f_{i,h}(K[w_1], \ldots, K[w_n]) \cdot \Delta^{2-h} \\ &= \sum_{h\in[0,2]} f_{i,h}(M[w_1] + \widetilde{w}_1 \cdot \Delta, \ldots, M[w_n] + \widetilde{w}_n \cdot \Delta) \cdot \Delta^{2-h} \\ &= g_i(\Delta) = A_{i,0} + A_{i,1} \cdot \Delta - e_i \cdot \Delta^2 \end{aligned}$$

where $A_{i,0}$ and $A_{i,1}$ denote the values that an honest $\mathcal{P}$ would compute. So B_i is the result of evaluating a quadratic polynomial at Δ.

By the exact same argument as in the proof of Theorem 14 (see Appendix F), we can can conclude theat no adversary can break the soundness of the protocol except with probability at most $2^{-\sigma}$.

Corrupted Verifier: The simulation of the verifier's view is straightforward: $\mathcal{S}$ simulates a party $\mathcal{V}^*$ in its head and gives control to $\mathcal{Z}$, and sends (corrupt, $\mathcal{V}$) to $\mathcal{F}_{\mathsf{ZK}}^k$. It also simulates an instance of $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$. We assume that the circuit $\mathcal{C}$ to prove is known.

1. Simulation of the preprocessing phase: $\mathcal{S}$ simulates the (Init) and (Extend, $n+1$) calls to $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$. Since $\mathcal{V}^*$ acts as the sender $\mathsf{P_S}$ towards $\mathcal{F}_{\mathsf{vole2k}}^{\ell,s}$, it is allowed to choose its outputs. For (Init), $\mathcal{S}$ receives $\Delta \in \mathbb{Z}_{2^s}$ from $\mathcal{V}^*$, and $K[\mu_i], K[o] \in \mathbb{Z}_{2^\ell}$ for $i \in [n]$ from $\mathcal{V}^*$.
2. To simulate the online phase, $\mathcal{S}$ proceeds as follows:
   - For every input $i \in [n]$ it sends a random $\delta_i \in_R \mathbb{Z}_{2^\ell}$ to $\mathcal{V}^*$, and computes $K[w_i] := K[\mu_i] - \delta_i \cdot \Delta$.
   - For the ith polynomial f_i, it computes B_i as the honest $\mathcal{V}$ would do in Step 2.
   - $\mathcal{S}$ receives $\boldsymbol{\chi} \in \mathbb{Z}_{2^s}^t$ from $\mathcal{V}^*$.
   - It computes $W := \sum_{i\in[t]} \chi_i \cdot B_i + B^* \in \mathbb{Z}_{2^\ell}$ where $B^* = K[o]$. Then it samples $V' \in_R \mathbb{Z}_{2^\ell}$ and sets $U' := W - V' \cdot \Delta$, and sends (U', V') to $\mathcal{V}^*$.

The view of $\mathcal{V}^*$ is distributed exactly as in the real execution of the protocol: The values δ_i are distributed uniformly in $\mathbb{Z}_{2^\ell}$ and therefore completely mask the inputs. Moreover, the values U, V computed by the honest $\mathcal{P}$ are also distributed uniformly at random due to the masking with A_0^* and A_1^*, respectively, under the condition that $W = U + V \cdot \Delta$ holds. Hence, the values U', V' sent to $\mathcal{V}^*$ by $\mathcal{S}$ are distributed identically, since we can recover the value W' that an honest $\mathcal{V}$ would compute. □