Actively Secure Arithmetic Computation and VOLE with Constant Computational Overhead*

Benny Applebaum† Niv Konstantini†

February 23, 2023

Abstract

We study the complexity of two-party secure arithmetic computation where the goal is to evaluate an arithmetic circuit over a finite field $\mathbb{F}$ in the presence of an active (aka malicious) adversary. In the passive setting, Applebaum et al. (Crypto 2017) constructed a protocol that only makes a *constant* (amortized) number of field operations per gate. This protocol uses the underlying field $\mathbb{F}$ as a black box, makes black-box use of (standard) oblivious transfer, and its security is based on arithmetic analogs of well-studied cryptographic assumptions. We present an actively-secure variant of this protocol that achieves, for the first time, all the above features. The protocol relies on the same assumptions and adds only a minor overhead in computation and communication.

Along the way, we construct a highly-efficient Vector Oblivious Linear Evaluation (VOLE) protocol and present several practical and theoretical optimizations, as well as a prototype implementation. Our most efficient variant can achieve an asymptotic rate of 1/4 (i.e., for vectors of length w we send roughly $4w$ elements of $\mathbb{F}$), which is only slightly worse than the passively-secure protocol whose rate is 1/3. The protocol seems to be practically competitive over fast networks, even for relatively small fields $\mathbb{F}$ and relatively short vectors. Specifically, our VOLE protocol has 3 rounds, and even for 10K-long vectors, it has an amortized cost per entry of less than 4 OT's and less than 300 arithmetic operations. Most of these operations (about 200) can be pre-processed locally in an offline non-interactive phase. (Better constants can be obtained for longer vectors.) Some of our optimizations rely on a novel intractability assumption regarding the non-malleability of noisy linear codes that may be of independent interest.

Our technical approach employs two new ingredients. First, we present a new information-theoretic construction of Conditional Disclosure of Secrets (CDS) and show how to use it in order to immunize the VOLE protocol of Applebaum et al. against active adversaries. Second, by using elementary properties of low-degree polynomials, we show that, for some simple arithmetic functionalities, one can easily upgrade Yao's garbled-circuit protocol to the active setting with a minor overhead while preserving the round complexity.

1 Introduction

Secure multiparty protocols (MPC) allow a set of parties to jointly compute a function over their inputs while keeping those inputs private. In many situations, the underlying sensitive data is *nu-*

*Supported by the Israel Science Foundation grant no. 2805/21. The conference version of this paper appears in Eurocrypt 2023.

†Tel-Aviv University, Israel {bennyap@post.tau.ac.il, NivKonst@gmail.com}.

merical, and the computation can be naturally expressed as a sequence of arithmetic operations such as addition, subtraction, and multiplication.[1] This calls for *secure arithmetic computation*, namely secure computation of functions defined by arithmetic operations. It is convenient to represent such a function by an *arithmetic circuit*, which is similar to a standard Boolean circuit except that gates are labeled by addition, subtraction, or multiplication. It is typically sufficient to consider such circuits that evaluate the operations over a large *finite field* $\mathbb{F}$, since arithmetic computations over the integers or (bounded precision) reals can be reduced to this case. Computing over finite fields (as opposed to integers or reals) can also be a feature, as it is useful for applications in threshold cryptography (see, e.g., [Gil99]).

It is always possible to reduce arithmetic computation to Boolean computation by implementing each arithmetic operation by Boolean circuit. However, this approach leads to a large blow-up in the circuit size.[2] Thus we strive for "purely arithmetic" solutions that avoid such an emulation. Specifically, following [ADI+17], we strive for a protocol that achieves a *constant computational overhead*. That is, we would like to securely evaluate any arithmetic circuit C over any finite field $\mathbb{F}$, with a computational cost (on a RAM machine) that is only a constant times bigger than the cost of performing $|C|$ field operations with no security at all. Here we make the standard convention of viewing the size of C also as a security parameter, namely the view of any adversary running in time $\mathsf{poly}(|C|)$ can be simulated up to a negligible error in $|C|$.

1.1 ADINZ: Constant Overhead with Passive Security

In [ADI+17] (hereafter referred to as ADINZ) it was shown that it is possible to realize 2-party arithmetic MPC with constant computational overhead in the presence of a *passive* (aka semi-honest) adversary. Specifically, ADINZ introduced the *Vector Oblivious Linear Evaluation* (VOLE) functionality in which the sender holds a pair of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^w$ and the receiver holds a scalar $x \in \mathbb{F}$ and gets as an output the vector $x\boldsymbol{a}+\boldsymbol{b}$, and showed how to (1) realize VOLE of width w with complexity of $O(w)$ and (2) how to use VOLE to realize *batch Oblivious Linear Evaluation* (batch-OLE) of length n with complexity $O(n)$. The latter functionality takes a pair of vectors $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{F}^n$ from the sender and a vector $\boldsymbol{y} \in \mathbb{F}^n$ from the receiver and delivers to the receiver the vector $\boldsymbol{y} \odot \boldsymbol{c}+\boldsymbol{d}$ where $\odot$ stands for entry-wise multiplication. Both the VOLE functionality and batch-OT functionality naturally extend the Oblivious Linear Evaluation (OLE) functionality [NP99, IPS09] that corresponds to the case where $w = 1$ or $n = 1$. Moreover, OLE, VOLE, and batch-OLE can be viewed as the arithmetic versions of oblivious transfer (OT), string-OT, and batch-OT, respectively. Indeed, just like in the binary setting, securely computing an ℓ-size arithmetic circuit reduces via an arithmetic-GMW construction [IPS09] to the task of securely computing batch-OLE of length ℓ. Based on this reduction, ADINZ constructed a constant-overhead MPC protocol for general arithmetic circuits.

The security of the ADINZ protocols is based on arithmetic analogs of well-studied cryptographic assumptions. Concretely, for the VOLE protocol, it suffices to assume the existence of a linear-time computable "code" over $\mathbb{F}$ for which noisy codewords are pseudorandom. Since a conservative choice of *constant-rate* noise suffices, one can instantiate this LPN-type assumption based on an arithmetic variant of Alekhnovich's assumption [Ale11] or based on the codes of Druk and Ishai [DI14]. The

[1] More complex numerical computations can typically be efficiently reduced to these simple ones, e.g., by using suitable low-degree approximations.

[2] For example, for the case of finite fields with n-bit elements, the size of the best known Boolean multiplication circuits is $\omega(n \log n)$.

batch-OLE protocol is based on an arithmetic version of a $\mathbf{NC^0}$ polynomial-stretch PRG [IKOS08, App16, AL16, AK19]. (See [Zic17] for security analysis of these two assumptions in the arithmetic setting.)

The ADINZ protocols also enjoy several useful properties. They make only black-box access of the field $\mathbb{F}$ and their arithmetic complexity (the number of field operations) grows linearly with the circuit size of the underlying functionality and is *independent* of the size of $\mathbb{F}$.[3] In addition, all protocols make a *black-box* use of a standard OT channel. In fact, in the hybrid-OT model, they achieve information-theoretic privacy against a corrupted VOLE/batch-OLE sender.[4] As advocated in [IPS08, IPS09], designing protocols in the OT-hybrid model yield several advantages such as native pre-processing [Bea96], simple amortization via OT-extension [Bea96, IKNP03, ALSZ15], and the ability to rely on different concrete implementations (including UC-secure ones) under a variety of computational or physical assumptions. Moreover, black-box usage is typically a necessary condition for obtaining practical efficiency. Indeed, the VOLE protocol of ADINZ makes only light-weight linear-algebraic operations and operates in a constant number of rounds, and a prototype implementation appeared in [ADI+17]. It should be mentioned that in the past few years the VOLE primitive has turned out to be an important building block with numerous applications such as secure computation of linear algebraic computations [MW08], round-efficient secure arithmetic computation via arithmetic garbling [AIK14], secure keyword search and set intersection [FIPR05, GN17], zero-knowledge proofs for arithmetic circuits [BCGI18, CDI+19, DIO21, WYKW21, BMRS21], and non-interactive secure computation [CDI+19, DIO21]. (See [IPS09, ADI+17, BCGI18] and references therein.)

1.2 Actively Secure Arithmetic MPC with Constant Overhead?

Unfortunately, the ADINZ protocols are only passively secure, and, as we will later see, an active adversary can completely break the privacy of both protocols (the VOLE protocol and the batch-OLE). One can probably construct an actively-secure protocol by combining ADINZ with constant-overhead arithmetic zero-knowledge proofs via an arithmetic version of the GMW-compiler [GMW87]. The elegant work of Bootle et al. [BCG+17] provides such a zero-knowledge protocol. However, this approach inherently makes a non-BB use of the underlying OT protocol. Also, the protocol of [BCG+17] has a super-constant round complexity.

For the special case of VOLE, the breakthrough results of Boyle et al. [BCGI18, BCG+19a] yield an actively-secure realization with constant overhead. However, their protocols are based on strong LPN-type assumptions with *sub-constant* noise rate. The protocol can be based on OT in a black-box way [BCG+19a] at the expense of further strengthening the underlying intractability assumption to a *leaky LPN* assumption and by making additional use of *correlation robust hash functions*.

To summarize, to the best of our knowledge, it is currently unknown how to realize batch-OT (or general arithmetic MPC) with active security, constant overhead, and black-box access to OT, regardless of the underlying assumption. For VOLE, the only known constructions either make a non-BB use of OT or rely on relatively strong intractability assumptions such as leaky-LPN with sub-constant noise and correlation robust hash functions. Our goal in this work is to avoid these limitations and derive an actively-secure arithmetic MPC protocol with constant overhead that

[3] The protocol additionally uses standard "bit-operations" whose complexity is dominated by the field operations.

[4] We mention that the aforementioned assumptions are not known to imply OT.

enjoys all the features of the ADINZ protocol.

2 Our Contribution

We resolve the above question in the affirmative by presenting actively-secure variants of the ADINZ protocols for VOLE, batch-OLE, and general arithmetic secure computation, that enjoy all the additional features and are based on the same assumptions. While our main focus is theoretical, we also present several practical optimizations to the VOLE protocol that make use of less conservative intractability assumptions. Details follow.

2.1 The VOLE protocols

Just like [ADI+17], we rely on the existence of *fast pseudorandom matrix* $M \in \mathbb{F}^{m \times k}$ where $m > k$ is a fixed polynomial in k (say $m = k^3$). Here "fast" means the mapping $u \mapsto M \cdot u$ can be computed by making only $O(m)$ arithmetic operations, and "pseudorandom" means that if we take a random vector in the image of M, and add a random μ-sparse noise vector to it, the resulting vector is computationally indistinguishable from a truly random vector over $\mathbb{F}^m$. The noise rate μ can be taken to be a constant, e.g., 1/4. There are several candidates for such fast pseudorandom matrices (see the discussion after Assumption 4.1). We prove the following theorem.

Theorem 2.1 (informal)**.** *Based on the fast pseudorandom matrix assumption, the VOLE functionality of width w can be realized with active security in the OT-hybrid model with arithmetic complexity of $O(w)$ and with perfect security against an active adversary that corrupts the Sender and computational security against the Receiver.*

This protocol (and all the protocols constructed in this work) makes black-box use of the underlying field and is therefore fully arithmetic in the sense of [IPS09]. (One can also derive the stronger form of arithmetic MPC of [AAB17] by instantiating the OT channel with an "arithmetic OT"). In addition, all the protocols that are constructed in this work admit a straight-line black-box simulator. In the 2-party setting, the existence of such simulators implies that the protocol is UC-secure as follows from [KLR10, Theorem 1.5] and [Lin03].

As already mentioned, Theorem 2.1 is the first to achieve constant overhead and black-box dependency in the OT based on a conservative constant-rate noise LPN-type assumption. Moreover, to the best of our knowledge, this is the first construction that achieves constant overhead and statistical Sender security in the OT-hybrid model, regardless of the underlying assumption,

Remark 1 (realizing the OT-channels with constant overhead)**.** *For the sake of communication/computational complexity, we charge every bit that is passed over the OT channel as a single bit/bit-operation. As already observed in [IKOS08, ADI+17], when each OT message is sufficiently long compared to the security parameter (which is the case in our protocols), such OT-channels can be realized securely based on an* arbitrary *OT protocol with the aid of a linear-time computable linear-stretch PRG. The existence of the latter follows from the binary version of the fast pseudorandom matrix assumption (see [AIK08, IKOS08, ADI+17]). In particular, by using any UC-secure OT protocol in the CRS setting (e.g., [PVW08]), we derive UC-secure implementations of our protocols in the standard model. (See Section 3.1 and Remark 2 for more details.)*

Optimizations: VOLE1 and VOLE2. Motivated by the rich applications of VOLE, we present several optimizations for the protocol. At the extreme, we present a VOLE protocol (VOLE1) that can achieve an asymptotic rate of 1/4 (i.e., the communication is dominated by sending roughly $4w$ elements of $\mathbb{F}$), which is only slightly worse than the passively-secure protocol whose rate is 1/3. Assuming that the OT consumes 2 rounds, VOLE1 has 3 rounds of computation. The protocol is provably secure against a computationally-unbounded sender, provably secure against a passive receiver, but only heuristically secure against an active receiver. That is, we conjecture that it achieves security against an active receiver, but do not have a security reduction to a clean intractability assumption. As a compromise, we introduce another protocol VOLE2 that slightly downgrades the asymptotic rate of 1/5, has 6 rounds, but can be proved secure based on a new, yet plausible, intractability assumption.[5]

The Correlated Noisy-Codeword Hardness Assumption. Our assumption intuitively asserts that given a random noisy codeword $\boldsymbol{c}$ sampled from a code T with noise pattern $\boldsymbol{e}$, it is hard to efficiently generate a new noisy codeword $\boldsymbol{d}$ whose noise pattern $\boldsymbol{e}'$ is non-trivially correlated with the noise pattern $\boldsymbol{e}$ in the following sense. The new noise vector $\boldsymbol{e}'$ is "far" from being a scalar multiple of $\boldsymbol{e}$ but it agrees with the original noise vector $\boldsymbol{e}$ with respect to the set of non-noisy coordinates $I = \{i : \boldsymbol{e}_i = 0\}$ of $\boldsymbol{e}$, i.e., $\boldsymbol{d}[I]$ is in the span of $T[I]$. Observe that such a noisy codeword can be generated by sampling a vector in the column span of $(T|\boldsymbol{c})$ and then modifying ℓ entries with the hope that all these entries fall out of the set of clean coordinates I. Such an attack succeeds with probability μ^ℓ, and our assumption states that one cannot do much better than this. (See Section 6 for a formal statement.) We believe that this assumption may be of independent interest and provide some evidence towards its validity in Section B.

Implementation and concrete complexity. The computational overhead of VOLE1 and VOLE2 are essentially the same. In Section 10, we analyze the concrete complexity of these protocols when instantiated with the same building blocks that were used in the passive setting of [ADI+17], suggest several practical optimizations, and present an implementation of the protocols. We show that the computational overhead compared to the passive version is minor (less than 20%). Furthermore, even for relatively short vectors of length 10000, our protocols have an amortized cost per VOLE entry of fewer than 4 OTs and less than 300 arithmetic operations (additions and multiplications). We use novel techniques (e.g., sparse LU-decomposition) to push most of these operations (about 200) to a *non-interactive* offline phase that can be pre-processed locally based only on the public parameters and local random tape. As a result, this preprocessing can be applied even before each party knows who will be her partner for the computation. The protocol is also very cheap for the receiver and requires in the online phase less than 10 arithmetic operations and 4 OT's per VOLE entry. (The sender's online amortized computation consists of 4 OT's and less than 80 arithmetic operations per entry.) Such a receiver-efficient protocol is especially useful for applications like VOLE-based zero-knowledge proofs (e.g., [DIO21]) in which the verifier plays the receiver and the prover plays the VOLE sender.

We believe that our protocol may be practically competitive over fast networks even for relatively small fields $\mathbb{F}$ and relatively short vectors. (Think for example, of an arithmetic zero-knowledge

[5] In fact, even our most conservative protocol (VOLE3) that proves Theorem 2.1 has an assymptotic rate of 1/5 and its amortized computational complexity is roughly the same. However, VOLE3 achieves this only over significantly longer vectors.

for a circuit that contains few 10K's gates, which, by [DIO21], translates into a single VOLE of comparable length.) When comparing our protocols to the alternative compressed-VOLE-based solution [BCGI18, BCG+19a], we see that the latter achieves a better rate of 1/2, but its amortization point seems to "kick in" only for longer vectors (due to the use of an "internal-MPC" protocol for securely realizing "short VOLE-correlations"). Thus, this approach is in a sense complementary to ours, and it will be interesting to study the combination of the two.[6] We also expect that additional optimizations of our implementation and the underlying building blocks will further improve the computational cost.

2.2 The batch-OLE protocol

The ADINZ [ADI+17] protocol for batch-OLE is based on the existence of *pseudorandom generator* (PRG) $G : \mathbb{F}^k \to \mathbb{F}^n$ with polynomial stretch, e.g., $n = k^2$, that is computable by a constant-depth ($\mathbf{NC^0}$) arithmetic circuit.[7] Candidate constructions are studied in [ADI+17, Zic17]. We prove the following theorem.

Theorem 2.2 (informal)**.** *Assuming the existence of an* $\mathbf{NC^0}$ *arithmetic PRG with polynomial stretch, the batch-OLE functionality of length n can be realized with active security in the VOLE-hybrid model with arithmetic complexity of $O(n)$ and by making a single call to an ideal k-length $O(n/k)$-width VOLE where the batch-OLE receiver (resp., sender) plays the role of the VOLE receiver (resp., sender). The protocol is perfectly secure against an active adversary that corrupts the receiver and computationally secure against an adversary that actively corrupts the sender. Moreover, the protocol has a constant number of rounds.*

Here the k-length $O(n/k)$-width VOLE functionality consists of k copies of $O(n/k)$-width VOLE. The protocol has 4 rounds (counting VOLE as a 2-round protocol). In [GIP+14] it is shown that the task of securely computing an arithmetic circuit C with active security reduces to batch-OLE of length $O(|C|)$ with constant computational overhead while preserving information-theoretic security. Combining this with Theorems 2.2 and 2.1, we derive the following corollary.

Corollary 2.3. *Assuming a fast pseudorandom matrix and an* $\mathbf{NC^0}$ *PRG with polynomial stretch, any two-party functionality that is computable by an arithmetic circuit C can be realized with arithmetic complexity of $O(|C|)$ in the OT-hybrid model while providing information-theoretic security for one party and computational security for the other party. The protocol makes black-box use of the underlying field.*

The corollary extends to any constant number of parties via standard reductions (e.g., [GIP+14]).

2.3 Technical Overview of the VOLE protocols

We briefly present some of the main technical ideas behind our constructions starting with the VOLE protocols.

[6] The current implementations of the compressed-VOLE-based solution are either restricted to the binary field [BCG+19a] or achieve passive security [SGRR19] and so we cannot compare the actual performance of our implementation against a compressed-VOLE-based implementation. Indeed, to the best of our knowledge, it seems that our work provides the first implementation of actively-secure VOLE over large fields.

[7] The exact level of stretch is not important since one can transform a given PRG with a polynomial stretch of $n = k^c$ for some $c > 1$, to a PRG with a stretch of $n = k^{c'}$ for an arbitrary constant $c' > c$ while increasing the depth of the circuit by a constant factor (see, e.g., [App16]).

The passive-VOLE protocol. The VOLE protocol of ADINZ17 is based on a protocol for "reverse VOLE" (RVOLE) functionality in which Bob holds a vector $\boldsymbol{a} \in \mathbb{F}^w$, Alice holds a vector $\boldsymbol{b} \in \mathbb{F}^w$ and a scalar $x \in \mathbb{F}$ and the goal is to deliver the value of $x\boldsymbol{a} + \boldsymbol{b}$ to Bob. Roughly, Bob sends to Alice an encryption $\boldsymbol{c} = E(\boldsymbol{a}) \in \mathbb{F}^m$ of $\boldsymbol{a}$ which is based on the fast pseudorandom matrix. (E also depends on some random field elements that are omitted for simplicity.) This encryption is "almost-linear" and so Alice can homomorphically compute a new ciphertext $\boldsymbol{d} = E(x\boldsymbol{a} + \boldsymbol{b}) \in \mathbb{F}^m$ by applying linear operations over $\boldsymbol{c}$. However, this ciphertext cannot be sent to Bob since it leaks information about x and $\boldsymbol{b}$. In particular, if Bob sees a coordinate of $\boldsymbol{d}$ that was "noisy" in the original ciphertext $\boldsymbol{c}$ he can efficiently extract the private input of Alice. (See Section 4.1.) To fix the problem, we let Bob read only the entries of $\boldsymbol{d}$ for which $\boldsymbol{c}$ is non-noisy.[8] Of course, Bob has to hide these locations, and so, for each entry $i \in [m]$, the parties invoke a standard 1-out-of-2 (or even all-or-nothing [Rab05]) OT-channel where Alice sends the pair $(\boldsymbol{d}_i, \perp)$ and Bob's selection bit determines whether to read $\boldsymbol{d}_i$ or to receive a $\perp$ symbol. While in the passive setting Bob can be trusted to read only the clean locations, an actively corrupt Bob can simply read all the entries of the vector $\boldsymbol{d}$, and completely recover Alice's input.

Securing the protocol against active Bob via CDS. Let us denote by T the matrix that corresponds to the linear part of the encryption E, i.e., T maps a plaintext of length w (and a vector of k random field elements) to a vector of length m whose noisy version corresponds to a ciphertext. Our first observation is that the above protocol remains secure if and only if the entries $I \subset [m]$ that Bob reads in the OTs satisfy the following condition: (*) The ciphertext $\boldsymbol{c}$ restricted to I is in the span of $T[I]$, the sub-matrix of T whose rows are indexed by I. (As a sanity check, observe that when Bob is honest the set I of "clean" coordinates satisfies the condition.) At this point, it is natural to try and extend the protocol with some form of zero-knowledge proof in which Bob proves that I satisfies the above condition. However, since I corresponds to Bob's input to the OTs (specifically, Bob's "selection bits") such a proof system seems to lead to a non-BB use of the OTs. To avoid this complication, we take an alternative route and make use of a special-tailored Conditional Disclosure of Secret (CDS) Protocol [GIKM00].

Roughly speaking, in such a protocol Alice chooses a random secret s and, for each index $i \in [m]$, Alice sends over the ith OT-call a pair of field elements $(\boldsymbol{d}_i, \boldsymbol{z}_i)$, and Bob has to choose whether to learn $\boldsymbol{d}_i$ or $\boldsymbol{z}_i$. For a selection vector I, Bob learns the vectors $(\boldsymbol{d}_i)_{i \in I}$ and $(\boldsymbol{z}_i)_{i \notin I}$. By design, the latter vector reveals the secret s if and only if I satisfies the (*) condition. Thus the CDS protocol effectively limits the query access to the OT's, and turns it into so-called a *generalized OT* (GOT) [IK97, Tas11, SSR08, GNN17].[9] Below, we present such a CDS protocol that achieves a constant computational overhead for both the sender and the receiver and information-theoretic security. Given such a protocol, we can immunize the RVOLE protocol by letting Alice re-encrypt her ciphertext $\boldsymbol{d}$ under the secret s. This approach yields only computational security since the key s, which is a single field element, is shorter than the vector $\boldsymbol{d}$. (A CDS with longer secrets would

[8] This information suffices to recover the plaintext $x\boldsymbol{a} + \boldsymbol{b}$ since the encryption internally employs a suitable error-correcting code. Indeed, [ADI+17] show how to combine a fast pseudorandom matrix with a linear-time error-correcting code and derive a linear-time encodable code that is pseudorandom under random noise but can be decoded in linear-time in the presence of random erasures.

[9] GOT allows Bob to retrieve a subset of the messages of Alice that are "authorized" according to some predicate P. Previous constructions were either based on decomposable randomized encoding (aka private-simultaneous messages protocols) [IK97] or on secret-sharing [Tas11, SSR08, GNN17]. We generalize these approaches by using CDS which is strictly weaker than both primitives.

lead to a super-constant computational overhead.) To achieve information-theoretic security, we note that it suffices to use s to encrypt the scalar x of the RVOLE protocol. That is, Alice invokes the modified RVOLE protocol (with the CDS mechanism) over the inputs $x + s$ and $\boldsymbol{b}$. We show that if Bob's selection strategy I satisfies the (*) condition, we can extract his inputs and perfectly simulate his view based on $x\boldsymbol{a} + \boldsymbol{b}$. If Bob's strategy I does not satisfy (*), he learns the vector $\boldsymbol{b}$ but x remains completely hidden, and we can perfectly simulate his view by sending $\boldsymbol{a} = 0^w$ to the ideal RVOLE functionality. We refer to the resulting protocol as the *modified-RVOLE* protocol (See Section 5).

Constructing the CDS. Our construction of the CDS is linear-algebraic in nature. As a starting point, we employ the following standard fact: Fix a matrix T and a vector $\boldsymbol{c}$. Suppose that we "encrypt" a secret s by sampling a random row vector $\boldsymbol{z}$ in the co-kernel of T, and publishing the "ciphertext" $s + \langle \boldsymbol{z}, \boldsymbol{c} \rangle$. If $\boldsymbol{c}$ is spanned by T the ciphertext equals to s, on the other hand, if $\boldsymbol{c}$ is *not* spanned by T, the ciphertext information theoretically hides s. Thus, *linear independence* is translated into *secrecy*. We can extend this idea to the CDS setting where we wish to reveal the secret iff $\boldsymbol{c}[I]$ is in the span of $T[I]$ for a subset I. To do this we reveal, for each $i \notin I$, the ith entry of our randomizer, $\boldsymbol{z}_i$, and send the ciphertext $s + \langle \boldsymbol{z}, \boldsymbol{c} \rangle$ in the clear. Given this information, one can map the ciphertext to $s + \langle \boldsymbol{z}[I], \boldsymbol{c}[I] \rangle$ which is decryptable if and only if $\boldsymbol{c}[I] \in \text{span}(T[I])$. While the above construction achieves privacy and correctness, it is not clear whether it achieves constant computational overhead. Indeed, we do not know how to sample a random vector in the co-kernel of T in linear time. Fortunately, there is a simple fix. To achieve a linear-time construction, we uniformly sample $\boldsymbol{z}$ from the entire space (without limiting to the co-kernel) and append to the CDS the value $\boldsymbol{z} \cdot T$ as a public value. Since right multiplication in T can be done in linear time, we can also left-multiply by T in linear time (following the "generalized transposition principle" [Bor57, IKOS08]) and so this variant can be realized with constant computational overhead. It is not hard to show that correctness and privacy still hold. (See Section 3.4 for a formal definition of CDS and Section 8 for details about the construction.)

Securing the protocol against active Alice? We move on and consider an actively-corrupted Alice. Clearly, even if Alice deviates from the protocol and does not compute the vector $\boldsymbol{d}$ properly, her view is still simulatable since all that she sees is a semantically-secure ciphertext $\boldsymbol{c}$. However, such misbehavior may lead Bob to abort and it is not fully clear how to simulate this case. Specifically, let us assume that Alice misbehaves and generates a vector $\boldsymbol{d} \notin \mathsf{colspan}\left(T|\boldsymbol{c}\right)$. The simulator detects this and can send an "abort" to the ideal functionality. However, in the real execution, Bob aborts only if his I-partial view is inconsistent, namely, if $\boldsymbol{d}[I] \notin \mathsf{colspan}\left((T|\boldsymbol{c})[I]\right)$ where $I = I(\boldsymbol{e})$ is the set of non-noisy coordinates in $\boldsymbol{e}$. To make the problem concrete, consider a malicious Alice that honestly computes $\boldsymbol{d}$ and then adds noise to the first coordinate of $\boldsymbol{d}$. In this case, the above simulator sends an abort, but in the real protocol, Bob aborts only if the first coordinate is in $I(\boldsymbol{e})$ which happens with constant probability $1 - \mu$. We present several solutions to this problem with different levels of efficiency.

1. The first solution is heuristic: We simply assume that the protocol is secure as it is. As evidence, we can prove this statement for the original RVOLE protocol (without the CDS) based on a variant of the Correlated Noisy-Codeword Hardness Assumption. (See Section A.) By using a straightforward reduction from VOLE to RVOLE [ADI+17], this leads to the VOLE1 protocol. (See Section 5.)

2. In the second solution, we first employ the modified-RVOLE protocol over random vectors $\boldsymbol{a}'$ and $\boldsymbol{b}'$ (this guarantees the ability to perfectly simulate an "abort" event), then use a small sub-protocol in which Alice proves that her CDS secret is independent of Bob's input (based on a simple commitment), and finally, we shift the vectors back to the real inputs vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ by exploiting the linearity of the VOLE functionality. To prove security we still need to extract Alice's input in the event that the protocol does not abort. For this, we rely on the aforementioned "Correlated Noisy-Codeword" intractability assumption. In a nutshell, we show that under this assumption, the simulator who is given a malformed ciphertext $\boldsymbol{d}$ either identifies that Bob would abort in the real execution or successfully extracts an effective input for Alice. Thus the security of the protocol can be based on the Correlated Noisy-Codeword assumption and on the fast pseudorandom matrix assumption. We refer to the resulting protocol as VOLE2 and note that it adds only a minor computational and communication overhead over RVOLE1. (See Section 6.)

3. Finally, our most conservative solution (VOLE3) relies solely on the fast pseudorandom matrix assumption. The starting point is again the modified-RVOLE protocol. We begin by observing that there exist efficient tests that determine whether Alice's ciphertext $\boldsymbol{d}$ is "valid" and whether Alice's vector of CDS messages $\boldsymbol{z}$ is "valid". Furthermore, when $\boldsymbol{d}$ and $\boldsymbol{z}$ are both valid, we can extract a unique effective input for Alice and properly simulate the protocol. We also note that there exists a strategy for Bob that detects (with probability 0.5) whether Alice cheats. Indeed, in the OT phase Bob can toss a coin and ask with probability $1/2$ to receive the vector $\boldsymbol{d}$ (by using $I = 1^m$) and with probability $1/2$ the vector $\boldsymbol{z}$ (by using $I = 0^m$) and check validity. When running in this "detection mode" Bob effectively gives up on the computation and just verifies whether Alice misbehaves or not. Note that Bob's decision is taken only in the OT phase and is hidden from Alice, and so effectively Alice first "commits" to strategy (cheat or not), and only then Bob decides whether to "call her bluff". Furthermore, even when Bob acts as a detector, we can fully simulate his view (since the protocol is actively-secure against any deviation of Bob). We will exploit these observations to obtain a "silent" cut-and-choose version of the protocol. Specifically, we realize W-width VOLE based on many calls to the modified-RVOLE protocol over shorter vectors of width $w \ll W$. Ignoring some technical details, we "sacrifice" a small fraction of these calls for cheating-detection[10], and glue together the remaining copies via a linear-time computable linear exposure-resilient function [CGH$^+$85] (also known as perfect deterministic extractors for bit-fixing sources). Such functions can be constructed based on linear-time encodable codes. (See Section 7.)

2.4 Technical Overview of the batch-OLE protocol

The ADINZ passive batch-OLE protocol relies on an arithmetic analog of Beaver's OT extension [Bea96]. Given an arithmetic PRG $G : \mathbb{F}^k \to \mathbb{F}^n$, the idea is to realize a pseudorandom batch-OLE in which Alice holds the vectors $\boldsymbol{c}$ and $\boldsymbol{d}$ of length n, Bob holds a seed of a PRG $\boldsymbol{x}$ of length k and the functionality stretches the seed $\boldsymbol{x}$ to a pseudorandom vector $\boldsymbol{y} = G(\boldsymbol{x})$ of length n, and delivers to Bob the value $\boldsymbol{y} \odot \boldsymbol{c} + \boldsymbol{d}$ where $\odot$ stands for entry-wise multiplication. The latter functionality f_G is realized by using an arithmetic variant of Yao's protocol. Specifically, Al-

[10] Interestingly, this detection is performed "silently": To test a session Bob just plays this session in a "detection mode". In contrast, in typical cut-and-choose-based solutions, Bob asks Alice to "open" a session. In fact, in our protocol we can even hide from Alice which sessions were tested by Bob.

ice prepares an arithmetic decomposable affine randomized encoding (DARE) [IK00, AIK06] (aka arithmetic garbled-circuit) of f_G and sends the "keys" that correspond to her entries. Bob recovers the keys of his inputs by making k calls to VOLE of width $w = O(n/k)$ and recovers the output. When the PRG is computable in $\mathbf{NC^0}$ the protocol can be realized with constant computational overhead. Clearly, the protocol is insecure in the presence of an actively corrupted Alice who can send a malformed encoding that corresponds to a different function. This well-known problem is extensively studied in the binary setting. We note that our concrete setting admits a simple and highly efficient solution.

Specifically, we strongly exploit the following properties: (1) We only care about the case where Bob's input $\boldsymbol{x}$ is chosen at random; (2) When Bob decodes the, possibly malformed, DARE (or the garbled circuit), each output of the computation can be written as a low-degree polynomial whose degree corresponds to the degree of f_G which is very small (constant) compared to the field size $|\mathbb{F}|$.[11] (3) The function f_G is linear in Alice's inputs.

Equipped with these observations, we run an extended variant of the passively-secure protocol in which Alice holds the pair $(\boldsymbol{c}, \boldsymbol{d})$ and another random pair of vectors $(\boldsymbol{c}', \boldsymbol{d}')$ of similar length, and Bob learns $\boldsymbol{y} \odot \boldsymbol{c} + \boldsymbol{d}$ and $\boldsymbol{y} \odot \boldsymbol{c}' + \boldsymbol{d}'$. Let us focus, for simplicity on the first output of f_G. By applying the decoder, Bob learns the values z_1 and z'_1 which are supposedly equal to $\boldsymbol{y}_1 \cdot \boldsymbol{c}_1 + \boldsymbol{d}_1$ and to $\boldsymbol{y}_1 \cdot \boldsymbol{c}'_1 + \boldsymbol{d}'_1$ where where $\boldsymbol{y}_1 = G(\boldsymbol{x})$. Assuming that Alice behaves properly, Bob can now compute the value of $\boldsymbol{y}_1 \cdot L(\boldsymbol{c}_1, \boldsymbol{c}'_1) + L(\boldsymbol{d}_1, \boldsymbol{d}'_1)$ for any linear combination L. The idea is to challenge Alice with a random non-trivial L and ask her to send $c = L(\boldsymbol{c}_1, \boldsymbol{c}'_1)$ and $d = L(\boldsymbol{d}_1, \boldsymbol{d}'_1)$, and let Bob check whether $L(z_1, z'_1) = c\boldsymbol{y}_1 + d$, and abort if the test fails.

First, observe that Alice's additional messages do not leak any information (since $\boldsymbol{c}'_1$ and $\boldsymbol{d}'_1$ mask the values of $\boldsymbol{c}_1$ and $\boldsymbol{d}_1$). Next, by using simple linear-algebraic arguments, we show that if Alice deviates from the protocol she will get caught except with probability $O(D/|\mathbb{F}|)$ where D is the degree of the PRG G. To see this, assume that Alice sends malformed garbled circuits for Bob's first outputs of f_G. This means that Bob computes $z_1 = Q(\boldsymbol{x})$ and $z'_1 = Q'(\boldsymbol{x})$ for some degree-D multivariate polynomials $Q(\cdot)$ and $Q'(\cdot)$ that are not both in the span of $\{G_1(\cdot), 1\}$ (e.g., there are no scalars $\boldsymbol{c}_1, \boldsymbol{d}_1$ for which $Q(\boldsymbol{x}) = \boldsymbol{c}_1 G(\boldsymbol{x}) + \boldsymbol{d}_1$). Consequently, if we take a random linear combination L of $Q(\cdot)$ and $Q'(\cdot)$, the resulting polynomial $L(Q, Q')$ almost surely falls out of the span of $\{G_1(\cdot), 1\}$. In this case, no matter how the scalars c, d are chosen by Alice, the polynomial $L(Q(\cdot), Q'(\cdot))$ will not be equal to the polynomial $cG_1(\cdot) + d$. Since both polynomials are of degree at most D, they will disagree over a random point $\boldsymbol{x}$, except with probability $D/|\mathbb{F}|$, and so Bob will almost surely catch the cheating. (See Section 9 for more details.)

As already mentioned this analysis crucially relies on the low-degree feature of the decoding procedure (property 2) to ensure that Q and Q' are of degree D. To the best of our knowledge, this is the first time that this feature is being employed.

Acknowledgement. We are grateful to Ivan Damgård and Yuval Ishai for early discussions that influenced this work. We also thank Yuval Ishai for explaining various aspects of [BCGI18, BCG+19a]. We thank the reviewers of Eurocrypt2023 for their comments.

Organization. Following some preliminaries in Section 3, we present in Section 4 the original ADINZ protocol, its underlying hardness assumption (Fast Pseudorandom Matrix), and analyze

[11] Indeed, here we assume that the field is sufficiently large. In contrast, the VOLE1 and VOLE2 protocols can be realized over small fields as well.

its (in)security in the malicious setting. In Section 5 we continue with a protocol for RVOLE that achieves active security against a corrupt receiver and use it to obtain VOLE1. The Correlated Noisy-Codeword hardness assumption is presented in Section 6 together with the VOLE2 protocol. Section 7 is devoted to VOLE3 and Section 8 is devoted to the CDS construction. The batch-OLE protocol appears in Section 9. In Section 10 we analyze the concrete efficiency of VOLE1 and present an implementation together with performance analysis. Appendix A shows that, under a variant of the Correlated Noisy-Codeword hardness assumption, the original ADINZ protocol achieves active security against malicious Alice, and Appendix B provides some evidence towards the validity of the Correlated Noisy-Codeword hardness assumption.

3 Preliminaries

Throughout this work, *efficient* adversaries are modeled by non-uniform polynomial-time algorithms.

3.1 Security model for malicious parties

Let $f = \left\{ f_k : \mathcal{A}_k \times \mathcal{B}_k \to \mathcal{O}_k^{\mathcal{A}} \times \mathcal{O}_k^{\mathcal{B}} \right\}_{k\in\mathbb{N}}$ be a two-party functionality that is parameterized by a security parameter k. We denote the protocol's parties by A (Alice) and B (Bob), and denote their inputs by $\boldsymbol{a}_k \in \mathcal{A}_k$ and $\boldsymbol{b}_k \in \mathcal{B}_k$, respectively. We define security against a malicious adversary via the standard REAL-IDEAL paradigm and consider, for simplicity the stand-alone model. Formally, we say that a two-party protocol $\mathcal{P}$ is *computationally secure* against Bob if for every efficient adversary B^* there exists an efficient simulator Sim, that (implicitly) takes the same non-uniform advice as B^*, such that, for any ensemble of inputs $\left\{(\boldsymbol{a}_k, \boldsymbol{b}_k)\right\}_{k\in\mathbb{N}}$ where $(\boldsymbol{a}_k, \boldsymbol{b}_k) \in \mathcal{A}_k \times \mathcal{B}_k$ the simulator *emulates* B^*, that is, the following ensembles (indexed by k) are computationally indistinguishable:

$$\textsc{Real}_{B^*,\mathcal{P}}(\boldsymbol{a}_k, \boldsymbol{b}_k) \quad \text{and} \quad \textsc{Ideal}_{\mathrm{SIM},f_k}(\boldsymbol{a}_k, \boldsymbol{b}_k).$$

Here the random variable $\textsc{Real}_{B^*,\mathcal{P}}(\boldsymbol{a}_k, \boldsymbol{b}_k)$ consists of the output of the adversary B^* (WLOG its view $\textsc{View}_{B^*}(\boldsymbol{a}_k, \boldsymbol{b}_k)$) concatenated with the output of the honest party A in a real execution of the protocol $\mathcal{P}$ over the inputs $(\boldsymbol{a}_k, \boldsymbol{b}_k)$. The random variable $\textsc{Ideal}_{\mathrm{SIM},f_k}(\boldsymbol{a}_k, \boldsymbol{b}_k)$ consists of the output of the simulator Sim concatenated with the output of the honest party A in an ideal execution that takes the following form: (1) Both parties send some inputs to the trusted party where Alice sends her input $\boldsymbol{a}_k$, and $\textsc{Sim}(\boldsymbol{b}_k)$ sends (a possibly modified) input $\boldsymbol{b}'_k$; (2) the trusted party computes $(y_A, y_B) = f_k(\boldsymbol{a}_k, \boldsymbol{b}'_k)$ and returns y_B to Sim; (3) The simulator sends to the trusted party either a "pass" message or an "abort" message according to which the trusted party either delivers y_A to A or sends her an "abort" message.[12] Security with respect to a malicious *Alice* is defined analogously.

The definition of statistical security is obtained by requiring that the statistical distance between the random variables $\textsc{Real}_{B^*,\mathcal{P}}(\boldsymbol{a}_k, \boldsymbol{b}_k)$ and $\textsc{Ideal}_{\mathrm{SIM},f_k}(\boldsymbol{a}_k, \boldsymbol{b}_k)$ is negligible in k. Perfect security corresponds to the case where the distance is 0. Finally, computational security (resp., statistical security, and perfect security) can be relaxed to *computational privacy* (resp., statistical privacy, and perfect privacy) by only requiring indistinguishably between the real view, $\textsc{View}_{B^*}(\boldsymbol{a}_k, \boldsymbol{b}_k)$, and the simulated view that is outputted by the simulator.

[12] Strictly speaking this does not guarantee passive security, however, it can be verified that all our protocols also satisfy passive security,

All our simulators are straight-line black-box simulators. In the two-party setting, the existence of such simulators implies that the protocol is UC-secure as follows from [KLR10, Theorem 1.5] and [Lin03].

The OT-hybrid Model. Our protocols are described in the OT-hybrid model, namely, in a model that allows parties to invoke in parallel multiple copies of an ideal oblivious transfer (OT) oracle [Rab05, EGL82, Gol04].[13] Formally, for a length parameter ℓ, the batch-OT oracle receives from the sender a vector of pairs of messages $(M_{i,0}, M_{i,1})_{i\in[\ell]}$ and the receiver sends a vector of bits $(b_i)_{i\in[\ell]}$. The oracle delivers to the receiver the values $(M_{i,b_i})_{i\in[\ell]}$. The messages $(M_{i,0}, M_{i,1})$ are typically taken to be field elements or k-bit strings for a security parameter k. For the sake of analysis, we measure the communication/computational complexity of such batch-OT as M bits and $O(M)$ bit-operations, respectively where $M = \sum_{i=1}^{\ell} |M_{i,0}| + |M_{i,1}|$ denotes the total bit-length of all the messages. This convention is supported by theoretical and practical instantiations of OT, as argued in Remark 2

Remark 2 (Realizing the OTs)**.** *As already observed in [IKOS08, ADI+17], when each OT message is sufficiently long compared to the security parameter, we can realize ℓ-OT with computational complexity (say circuit size) of $O(M)$ and communication complexity of M bits where $M = \sum_{i=1}^{\ell} |M_{i,0}| + |M_{i,1}|$ denotes the total bit-length of all the OT messages. This can be done based on an arbitrary actively-secure OT protocol and a linear-time PRG with linear stretch as follows. First, make $\ell(k)$ calls to the "base" OT to deliver ℓ pairs of k-bit random seeds $(m_{i,0}, m_{i,1})$ as messages, and then send in the clear the encrypted messages $M_{i,b} \oplus G_{i,b}(m_{i,b})$ for $i \in [\ell]$ and $b \in \{0,1\}$, where $G_{i,b}$ expands the k-bit seed to a string of length $|M_{i,b}|$ in time $O(|M_{i,n}|)$. Assuming that each call to the base OT has a cost of $T(k)$-time and $C(k)$-communication, the overall computational (resp., communication) complexity is $\ell \cdot T + O(M)$ (resp., $\ell \cdot C + M$) which, for sufficiently long messages yield the required result. Finally, we note that such a PRG, $G_{i,b}$, can be constructed based on the existence of a linear-stretch pseudorandom generator $G : \{0,1\}^k \to \{0,1\}^{2k}$ which is also computable in linear-time $O(k)$ [IKOS08]. The latter assumption is a relatively mild one and it follows from the binary version of our main intractability assumption (Assumption 4.1), see [AIK08, IKOS08, ADI+17]. Finally, in practice, we can realize an actively-secure batch-OT very efficiently via OT extension [Bea96, IKNP03, KOS15, ALSZ15, KK13, BCG+19b] with a communication cost of $M + O(k)$ and while performing a constant number of symmetric operations per OT (see Section 10).*

3.2 Linear algebraic notations

We define some linear-algebraic notation. Below $\mathbb{F}$ denotes some finite field and $m \in \mathbb{N}$ is a positive integer.

Selective matrix-vector entries. For a vector $\boldsymbol{d} \in \mathbb{F}^m$ and a 0-1 vector $I = (I_1, \ldots, I_m) \in \{0,1\}^m$, we define the vector $\boldsymbol{d}[I] \in \mathbb{F}^m$ such that its ith entry is d_i if $I_i = 1$ and 0 otherwise. This notation can be used for both row and column vectors and can be naturally extended to matrices as follows. For a $m \times k$ matrix M whose rows are denoted by $M_1, \ldots, M_m \in \mathbb{F}^k$, and a binary vector $I \in \{0,1\}^m$, we let $M[I] \in \mathbb{F}^{m\times k}$ denote the matrix whose ith row is M_i if

[13] In fact, we only need such a channel in one direction from the VOLE receiver Alice to the VOLE sender BOb.

$I_i = 1$ and 0^k otherwise. Note that this operator is linear and can be written in the following matrix form:

$$\boldsymbol{d}[I] = \begin{pmatrix} I_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & I_m \end{pmatrix} \cdot \boldsymbol{d} \qquad M[I] = \begin{pmatrix} I_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & I_m \end{pmatrix} \cdot M$$

Vector of clean coordinates. Let $\boldsymbol{e} \in \mathbb{F}^m$ denote a vector (typically viewed as a noise vector). We define the vector $I(\boldsymbol{e}) \in \{0,1\}^m$ such that its ith entry is 1 if and only if $\boldsymbol{e}_i = 0$. Note that by our notations $\boldsymbol{e}[I(\boldsymbol{e})] = 0^m$.

Matrix-vector concatenation. Let M be a matrix of dimensions $m \times k$ and a vector $\boldsymbol{c} \in \mathbb{F}^m$. We define $M|\boldsymbol{c}$ to be the result matrix that is obtained by concatenating the column vector $\boldsymbol{c}$ to the matrix M from the right side.

Bernoulli vector. Let $\mathsf{BER}^m(p)$ for real number $p \in [0,1]$ be the distribution of binary vectors $I = (I_1, ..., I_m)$ of length m with i.i.d entries such that for any i: I_i takes a value of 1 with probability p.

3.3 Oblivious Linear Evaluation and friends

We will be particularly interested in the following arithmetic functionalities: The OLE functionality over a finite field $\mathbb{F}$ takes an input $x \in \mathbb{F}$ from Alice and a pair $a, b \in \mathbb{F}$ from Bob and delivers $ax + b$ to Alice. VOLE of width w is similar, except that the input of Bob is a pair of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^w$ and Alice's output is the vector $\boldsymbol{a}x + \boldsymbol{b}$. OLE and VOLE can be viewed as arithmetic analogs of bit-OT and string-OT, respectively. Indeed, in the case $\mathbb{F} = \mathbb{F}_2$, the OLE functionalities coincide with the corresponding OT functionalities up to a local relabeling of the inputs. In the same manner, we can define the RVOLE functionality, where Bob has a single input $\boldsymbol{a} \in \mathbb{F}^w$, while Alice gets an additional input $\boldsymbol{b} \in \mathbb{F}^w$ and the functionality outputs nothing to Alice and the vector $\boldsymbol{a}x + \boldsymbol{b}$ to Bob.

Remark 3 (from RVOLE to VOLE)**.** *One can turn RVOLE to VOLE via the following simple reduction from [ADI+17]. Given an input $x \in \mathbb{F}$ for Alice, and inputs $\boldsymbol{a} \in \mathbb{F}^w$ and also $\boldsymbol{b} \in \mathbb{F}^w$ for Bob the parties continue as follows:*

1. ***Alice and Bob:*** *Invoke a protocol for RVOLE functionality, where Bob uses his input $\boldsymbol{a} \in \mathbb{F}^w$ and Alice uses her input x and a randomly chosen input $\boldsymbol{b}' \in \mathbb{F}^w$. Resulting in the vector $\boldsymbol{v} = \boldsymbol{a}x + \boldsymbol{b}'$ held by Bob.*
2. ***Bob:*** *Sends $\boldsymbol{u} = \boldsymbol{v} + \boldsymbol{b}$ to Alice.*
3. ***Alice*** *Outputs the vector $\boldsymbol{u} - \boldsymbol{b}' = \boldsymbol{a}x + \boldsymbol{b}' + \boldsymbol{b} - \boldsymbol{b}' = \boldsymbol{a}x + \boldsymbol{b}$.*

It is not hard to verify that this protocol forms a perfectly active reduction from VOLE to RVOLE with a communication overhead of additional w field elements and computational overhead of w additions and w subtraction.

Family of finite fields. We always assume that our functionalities are implicitly parameterized by a family of finite fields whose size may grow with the security parameter. Throughout the paper, we fix this family $\mathbb{F} = \{\mathbb{F}_k\}_{k\in\mathbb{N}}$ and assume that it is *efficiently computable*, that is, one should be able to compute all field operations in $\mathsf{poly}(k)$ time (including the ability to add/subtract/multiply/divide and to sample a random field element). Note that this requirement implies that $|\mathbb{F}_k| \leq 2^{\mathsf{poly}(k)}$ and so field elements can be represented by $\mathsf{poly}(k)$-bit strings. In fact, for our protocols, we only need black-box access to the field operations, and the ability to send field elements either directly or over an OT channel. By default, we also assume that the field is sufficiently large, e.g., exponentially large in the security parameter. Specifically, for an appropriate choice of the width parameter w (e.g., cubic in k) our protocols for width-w VOLE require $O(w)$ field operations, and at most $O(w \log |\mathbb{F}_k|)$ Boolean operations.[14] The overall complexity is therefore dominated by the arithmetic complexity $O(w)$ which is optimal. Indeed, even in an insecure implementation, w arithmetic operations are needed for w-width VOLE.

It should be mentioned that the assumption regarding the field size is mainly needed for achieving linear-time efficiency and our protocols (or close variants of them) remain secure even when the field is of small constant size (the error is always negligible in the security parameter). See Remark 4. We also mention that all the protocols in this work can be modeled as "protocol compilers" similarly to [ADI+17].

3.4 Conditional Disclosure of Secrets Protocol

The following variant of Conditional Disclosure of Secrets (CDS) protocols [GIKM00] will be employed in our protocols. We consider a model where $m+1$ servers $Q_0, Q_1, \ldots, Q_m$ hold a secret s and a common random string r. In addition, every server Q_i for $i > 0$ holds an input x_i of some m-input predicate function f. The server Q_0 holds no input and is sometimes referred to as an *offline server*. In a CDS protocol for f, for every $i \in [m]$, server Q_i sends a message to a referee, based on r, s, and x_i, while server Q_0 sends a message that depends only on the secret and the randomness, such that the referee, which holds the inputs $x_1, \ldots, x_m$ and sees the messages from the servers, can reconstruct the secret s if $f(x_1, \ldots, x_m) = 1$, and it cannot learn any information about the secret s if $f(x_1, \ldots, x_m) = 0$.

Definition 3.1 (Conditional disclosure of secrets protocols)**.** *Let $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_m \to \{0,1\}$ be a m-input function. A $m+1$-server CDS protocol $\mathcal{P}$ for f, with domain of secrets $\mathcal{S}$, domain of common random strings $\mathcal{R}$, and finite message domains $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_m$, consists of $m+1$ deterministic message computation functions $\mathsf{Enc}_0, \mathsf{Enc}_1, \ldots, \mathsf{Enc}_m$, where $\mathsf{Enc}_i : \mathcal{X}_i \times \mathcal{S} \times \mathcal{R} \to \mathcal{M}_i$ for every $i \in [m]$, and $\mathsf{Enc}_0 : \mathcal{R} \to \mathcal{M}_0$. For an input $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$, secret $s \in \mathcal{S}$, and randomness $r \in \mathcal{R}$, we let $\mathsf{Enc}(\boldsymbol{x}, s; r) = (\mathsf{Enc}_0(r), \mathsf{Enc}_1(x_1, s; r), \ldots, \mathsf{Enc}_m(x_m, s; r))$. We say that a protocol $\mathcal{P}$ is a CDS protocol for f if it satisfies the following properties:*

Perfect correctness. *There is a deterministic reconstruction function $\mathsf{Dec} : \mathcal{X}_1 \times \cdots \times \mathcal{X}_m \times \mathcal{M}_0 \times \cdots \times \mathcal{M}_m \to \mathcal{S}$ such that for every input $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$ for which $f(x_1, \ldots, x_m) = 1$, every secret $s \in \mathcal{S}$, and every common random string $r \in \mathcal{R}$, it holds that $\mathsf{Dec}(\boldsymbol{x}, \mathsf{Enc}(\boldsymbol{x}, s; r)) = s$.*

Perfect privacy. *For every input $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$ for which $f(x_1, \ldots, x_m) = 0$*

[14] Formally, we measure the complexity in the arithmetic RAM model as in [IPS09, ADI+17].

and every pair of secrets $s, s' \in \mathcal{S}$ it holds that $\mathsf{Enc}(\boldsymbol{x}, s; r) \equiv \mathsf{Enc}(\boldsymbol{x}, s'; r)$, where r is sampled with uniform distribution from $\mathcal{R}$.

In the special case where all the "zero-input messages" $\mathsf{Enc}_1(0, s; r), \ldots, \mathsf{Enc}_m(0, s; r)$ are empty strings (regardless of s, r), the CDS is actually a secret-sharing scheme for the access structure given by the function f. Similarly, when all the "one-input messages" $\mathsf{Enc}_1(1, s; r), \ldots, \mathsf{Enc}_m(1, s; r)$ are empty strings, the CDS forms a secret-sharing scheme for the access structure given by the function $\neg f$. Jumping ahead, our CDS will be of the latter type. (Though most of our constructions could be based on an arbitrary CDS.)

3.5 Decomposable affine randomized encoding (DARE)

Let $f : \mathbb{F}^\ell \to \mathbb{F}^m$ where $\mathbb{F}$ is some finite field.[15] We say that a function $\hat{f} : \mathbb{F}^\ell \times \mathbb{F}^\rho \to \mathbb{F}^m$ is a *perfect randomized encoding* [IK00, AIK06] of f if for every input $x \in \mathbb{F}^\ell$, the distribution $\hat{f}(x; r)$ induced by a uniform choice of $r \leftarrow \mathbb{F}^\rho$, "encodes" the string $f(x)$ in the following sense:

1. (Correctness) There exists a decoding algorithm Dec such that for every $x \in \mathbb{F}^\ell$ it holds that
$$\Pr_{r \leftarrow \mathbb{F}^\rho}[\mathsf{Dec}(\hat{f}(x; r)) = f(x)] = 1.$$
2. (Privacy) There exists a randomized algorithm $\textsc{Sim}$ such that for every $x \in \mathbb{F}^\ell$ and uniformly chosen $r \leftarrow \mathbb{F}^\rho$ it holds that
$$\textsc{Sim}(f(x)) \quad \text{is distributed identically to} \quad \hat{f}(x; r).$$

We say that $\hat{f}(x; r)$ is decomposable and affine if $\hat{f}$ can be written as

$$\hat{f}(x; r) = (\hat{f}_0(r), \hat{f}_1(x_1; r), \ldots, \hat{f}_n(x_\ell; r))$$

where $\hat{f}_i$ is linear in x_i, i.e., it can be written as $\vec{a}_i x_i + \vec{b}_i$ where the vectors $\vec{a}_i$ and $\vec{b}_i$ arbitrarily depend on the randomness r.

It follows from [IK02] (cf. [AIK14]) that every single-output function $f : \mathbb{F}^d \to \mathbb{F}$ which can be computed by a constant-depth circuit (aka $\mathbf{NC^0}$ function) admits a decomposable encoding that can be encoded and decoded by an arithmetic circuit of finite complexity C which depends only in the circuit depth. Furthermore, the algebraic degree of the decoder $\mathsf{Dec} : \mathbb{F}^m \to \mathbb{F}$ equals the algebraic degree of f. Note that any multi-output function can be encoded by concatenating independent randomized encodings of the functions defined by its output bits. Thus, we have the following:

Fact 3.2. *Let $f : \mathbb{F}^\ell \to \mathbb{F}^m$ be an $\mathbf{NC^0}$ function. Then, f has a DARE $\hat{f}$ which can be encoded, decoded, and simulated by an arithmetic circuit of size $O(m)$ where the constant in the big-O notation depends on the circuit depth.[16] Furthermore, the degree of the decoder equals the degree of f.*

We mention that the circuits for the encoding, decoder, and simulator can be all constructed efficiently given the circuit for f.

[15] The following actually holds even for the case of general rings.

[16] This hidden constant corresponds to the maximal complexity of encoding a single output of f. The latter is at most cubic in the size of the branching program that computes f_i (and can be even smaller for some concrete useful special cases).

4 The ADINZ protocol

The ADINZ [ADI+17] protocol for VOLE is based on a gadget ("encoder") that allows fast encoding and decoding under erasures but semantically hides the encoded messages in the presence of noise. This gadget is mainly based on a public matrix $M \in \mathbb{F}_k^{m \times k}$ with the following (LPN-style) pseudorandomness property: If we take a random vector in the image of M, and add a sparse noise to it, the resulting vector is computationally indistinguishable from a truly random vector over $\mathbb{F}_k^{m(k)}$. The noise distribution that is being used in the ADINZ protocol corresponds to an additive noise vector $\boldsymbol{e} \in \mathbb{F}_k^{m(k)}$ where each coordinate of $\boldsymbol{e}$ is assigned independently with the value of zero with probability $1-\mu$ and with a uniformly chosen non-zero element from $\mathbb{F}_k$ with probability μ. We let $\mathcal{D}(\mathbb{F}_k)_\mu^m$ denote the corresponding noise distributions for such vectors of length m. For concreteness, the reader may think of μ as a small constant, say $1/4$, however μ can also be chosen so that it tends to 0 when the security parameter k tends to infinity. The properties of the ADINZ gadget are summarized in the following assumption.

Assumption 4.1. ***(Fast pseudorandom matrix)*** *There exists a noise rate $\mu = \mu(k) < 1/2$ and an efficient randomized algorithm $\mathcal{M}$ that given a security parameter 1^k and a fields family representation $\mathbb{F}$, samples a $m \times k$ ($m = O(k^3)$) matrix M over $\mathbb{F}_k$ such that the following holds:*

1. *(Linear-time computation) The mapping $f_M : \boldsymbol{r} \to M\boldsymbol{r}$ can be computed in time that linear in the output length m, i.e., by performing $O(m)$ arithmetic operations.*
2. *(Noisy-codeword is pseudorandom) The following ensembles are computationally indistinguishable:*
$$\left\{(M, M\boldsymbol{r} + \boldsymbol{e})\right\}_{k\in\mathbb{N}} \approx_c \left\{(M, \boldsymbol{u})\right\}_{k\in\mathbb{N}}$$
where $M \leftarrow \mathcal{M}(1^k, \mathbb{F}), \boldsymbol{r} \leftarrow \mathbb{F}_k^k, \boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$ and $\boldsymbol{u} \leftarrow \mathbb{F}_k^m$.
3. *(Linear independence) If we sample $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$ and keep each of the first $u = O(k \log^2 k)$ rows independently with probability $1-\mu$ (and remove all other rows), then, except with negligible probability in k, the resulting matrix has full rank of k.*

Concrete instantiations of this matrix-ensemble $\mathcal{M}$ (e.g., based on sparse matrices or on the Druk-Ishai ensemble [DI14]) are discussed in [ADI+17]. The ADINZ encoder also makes use of a (non-cryptographic) linear error correcting code $\mathsf{Ecc} : \mathbb{F}_k^w \to \mathbb{F}_k^v$ which encodes vectors of length w into vectors of length v over the field $\mathbb{F}_k$, with constant rate R and linear time encoding and decoding, such that decoding is possible with high success probability from a constant fraction of erasures μ' which is slightly larger than the noise rate μ. (For $\mu = \frac{1}{4}$ we can take $\mu' = \frac{1}{3}$). Such codes are known to exist and can be efficiently constructed given black-box access to $\mathbb{F}_k$.
The code Ecc and the matrix M are combined together into the so-called protocol's encoder:

ADINZ Protocol's encoder. Given $k \in \mathbb{N}$, $m = O(k^3)$, $w = O(k^3)$, $\boldsymbol{r} \in \mathbb{F}_k^k$ and $\boldsymbol{a} \in \mathbb{F}_k^w$, let M be a $m \times k$ fast pseudorandom matrix. We define the encoding gadget $E_{\boldsymbol{r}}(\boldsymbol{a})$ to be:

$$E_{\boldsymbol{r}}(\boldsymbol{a}) = M \cdot \boldsymbol{r} + 0^u \circ \mathsf{Ecc}(\boldsymbol{a})$$

where $\mathsf{Ecc} : \mathbb{F}_k^w \to \mathbb{F}_k^v$, $u = 2k \log^2 k$, $v = m - u$ and $\circ$ denotes concatenation (so $0^u \circ \mathsf{Ecc}(\boldsymbol{a})$ is a vector of length m). Equivalently, for an information vector $\boldsymbol{a} \in \mathbb{F}_k^w$ and randomness vector $\boldsymbol{r} \in \mathbb{F}_k^k$, we can write the encoder as

$$E_{\boldsymbol{r}}(\boldsymbol{a}) = T \cdot \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{a} \end{pmatrix},$$

where the encoder matrix is

$$T = \left(\begin{array}{c|c} & \mathbf{0}_{u\times w} \\ M_{m\times k} & \\ & \mathsf{Ecc}_{v\times w} \end{array} \right) \tag{1}$$

and $\mathsf{Ecc}_{v\times w} \in \mathbb{F}_k^{v\times w}$ is the generating matrix of the error correcting code. By exploiting Assumption 4.1 and the features of the error correcting code, the encoder E satisfies the following properties:

1. (Fast and Linear) The mapping $E_{\boldsymbol{r}}(\boldsymbol{a})$ can be computed by making only $O(m)$ arithmetic operations. Moreover, it is a linear function of $\boldsymbol{r}$ and $\boldsymbol{a}$ and so $E_{\boldsymbol{r}}(\boldsymbol{a})+E_{\boldsymbol{r}'}(\boldsymbol{a}') = E_{\boldsymbol{r}+\boldsymbol{r}'}(\boldsymbol{a}+\boldsymbol{a}')$.
2. (Hiding under errors) For any message $\boldsymbol{a} \in \mathbb{F}_k^w$ and $\boldsymbol{r} \leftarrow \mathbb{F}_k^k, \boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_m^\mu$ the vector $E_{\boldsymbol{r}}(\boldsymbol{a}) + \boldsymbol{e}$ is pseudorandom. Namely: for any ensemble $\{\boldsymbol{a}_k\}_{k\in\mathbb{N}}$ the following ensembles are computationally indistinguishable:

$$\Big\{\big(M, E_{\boldsymbol{r}}(\boldsymbol{a}_k) + \boldsymbol{e}\big)\Big\}_{k\in\mathbb{N}} \approx_c \Big\{(M, \boldsymbol{u})\Big\}_{k\in\mathbb{N}}$$

   where $M \leftarrow \mathcal{M}(1^k, \mathbb{F}), \boldsymbol{r} \leftarrow \mathbb{F}_k^k, \boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$ and $\boldsymbol{u} \leftarrow \mathbb{F}_k^m$. In particular, a noisy codeword computationally "hides" $\boldsymbol{a}$.
3. (Fast decoding under erasures) Given a random vector $I \leftarrow \mathsf{BER}^m(1-\mu)$ and a code $\boldsymbol{d}[I] = E_{\boldsymbol{r}}(\boldsymbol{a})[I]$ (i.e. each coordinate is erased independently with probability μ) it is possible to recover the vector $\boldsymbol{a}$, with negligible error probability, by making only $O(m)$ arithmetic operations. We first recover $\boldsymbol{r}$ by solving the linear system $\boldsymbol{d}_{\text{top}}[I_{\text{top}}] = M_{\text{top}}[I_{\text{top}}]\boldsymbol{r}$ (where "top" means top u coordinates) via Gaussian elimination in $O(m)$ arithmetic operations. By Assumption 4.1 (property 3) the system is likely to have a unique solution. Then we compute $M[I_{\text{bot}}]\boldsymbol{r}$ in time $O(m)$, subtract from $\boldsymbol{d}[I_{\text{bot}}]$ to get the vector $\mathsf{Ecc}(\boldsymbol{a}))[I_{\text{bot}}]$ and recover $\boldsymbol{a}$ by erasure decoding in time $O(m)$.

Remark 4 (On the choice of parameters)**.** *Some of the above requirements are tailored to achieve an asymptotic computational complexity of $O(w)$ field operations. This includes the choice of the values of m, w and u, the requirements for "fast" computation of E and "fast" decoding under erasures, and the assumption that the field size is exponential in the security parameter. All these requirements can be waived without affecting the security of the protocol. (Assuming that the pseudorandomness assumption holds.) In particular, for concrete settings, it may be better to set these parameters differently as done in Section 10.*

The ADINZ protocol for RVOLE with honest parties appears as Protocol 1.

Protocol 1 (**ADINZ RVOLE protocol**). *To initialize the protocol Bob samples the matrix* $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$ *and sends it to Alice.*

1. ***Bob:*** *Given an input* $\boldsymbol{a} \in \mathbb{F}_k^w$*, Bob samples vectors* $\boldsymbol{r} \leftarrow \mathbb{F}_k^k$*,* $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$*, set the vector* $I = I(\boldsymbol{e})$ *and sends the vector:* $\boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}) + \boldsymbol{e}$ *to Alice.*
2. ***Alice:*** *Given the inputs* $\boldsymbol{b} \in \mathbb{F}_k^w$*,* $x \in \mathbb{F}_k$*, and Bob's message* $\boldsymbol{c}$*, Alice samples a vector:* $\boldsymbol{r'} \leftarrow \mathbb{F}_k^k$ *and computes the vector* $\boldsymbol{d} = x\boldsymbol{c} + E_{\boldsymbol{r'}}(\boldsymbol{b})$*.*
3. ***Alice and Bob:*** *Invoke* m*-batch OT where the* i*th entry of Alice is* $(\boldsymbol{d}, \perp)$ *and Bob uses the vector* I *as its selection vector.*
4. ***Bob:*** *Gets the vector* $\boldsymbol{d}[I]$ *that can be written as:*

$$\boldsymbol{d}[I] = E_{x\boldsymbol{r}+\boldsymbol{r'}}(x\boldsymbol{a} + \boldsymbol{b})[I] + x\boldsymbol{e}[I]$$

and since that $I = I(\boldsymbol{e})$ *Bob has just received the vector:*

$$\boldsymbol{d}[I] = E_{x\boldsymbol{r}+\boldsymbol{r'}}(x\boldsymbol{a} + \boldsymbol{b})[I]$$

Using the fast-decoding property of the gadget E *(property 3), Bob recovers the vector* $x\boldsymbol{a} + \boldsymbol{b}$ *(by making* $O(m)$ *arithmetic operations) and outputs it.*

Remark 5 (About the set-up). *In this protocol (and all the subsequent ones) the set-up step in which the matrix is sampled can be done once and for all. This is reflected in the security proofs which work even if Alice's simulator receives* M *as an external input.*

4.1 Insecurity against a malicious Bob

Suppose that Alice is honest, then at the end of the OT phase in the ADINZ protocol Bob holds the vector $\boldsymbol{d}[I] \in \mathbb{F}^m$ where $\boldsymbol{d} = x\boldsymbol{c} + E_{\boldsymbol{r'}}(\boldsymbol{b})$, namely:

$$\boldsymbol{d}[I] = E_{\boldsymbol{r'}}(\boldsymbol{b})[I] + x\boldsymbol{c}[I] = T[I] \begin{pmatrix} \boldsymbol{r'} \\ \boldsymbol{b} \end{pmatrix} + x\boldsymbol{c}[I].$$

As shown in [ADI+17], when Bob honestly follows the protocol, this value leaks no non-trivial information about Alice's input (except for the output of the functionality). The proof relies on the fact that the vector $\boldsymbol{c}$ is a noisy codeword and the binary vector of OT choices, I, is taken to be the indicator vector of the non-noisy coordinates, and so $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$. Of course, when Bob actively deviates from the protocol he may choose his vectors $(\boldsymbol{c}, I) \in \mathbb{F}_k^m \times \left\{0,1\right\}^m$ such that $\boldsymbol{c}[I] \notin \mathsf{colspan}\left(T[I]\right)$. In the following claim we show that in this case Bob can extract Alice's input x.

Claim 4.2. *Fix a matrix* $T \in \mathbb{F}^{m \times n}$*, and vectors* $\boldsymbol{c} \in \mathbb{F}^m$*,* $I \in \left\{0,1\right\}^m$ *such that* $\boldsymbol{c}[I] \notin \mathsf{colspan}\left(T[I]\right)$*. Let* $\boldsymbol{d}$ *be a vector that is computed honestly by Alice, namely:* $\boldsymbol{d} = T \begin{pmatrix} \boldsymbol{r'} \\ \boldsymbol{b} \end{pmatrix} + x\boldsymbol{c}$ *for some* $\left(\boldsymbol{r'}, \boldsymbol{b}\right) \in \mathbb{F}_k^k \times \mathbb{F}_k^w$*. Then, given* $(T, \boldsymbol{c}, I, \boldsymbol{d}[I])$*, it is possible to efficiently recover Alice's input* x*.*

Proof of claim. Since $\boldsymbol{c}[I] \notin \mathsf{colspan}\left(T[I]\right)$, we can find a vector $\boldsymbol{q} \in \mathsf{kernel}\left((T[I])^\intercal\right)$ such that

$\boldsymbol{q}^\intercal \cdot \boldsymbol{c}[I] = 1$ and compute the value

$$\boldsymbol{q}^\intercal \cdot \boldsymbol{d}[I] = \mathbf{0}^n \cdot \begin{pmatrix} \boldsymbol{r}' \\ \boldsymbol{b} \end{pmatrix} + x\boldsymbol{q}^\intercal \cdot \boldsymbol{c}[I] = x\boldsymbol{q}^\intercal \cdot \boldsymbol{c}[I] = x,$$

as required. □

We conclude that one cannot simulate the view of a malicious Bob that in the OT-phase uses a selection vector I for which $\boldsymbol{c}[I] \notin \mathsf{colspan}\left(T[I]\right)$. Consequently, the original protocol is insecure in this setting. On the positive side, we will later see that the above attack is the "only possible" attack that can be mounted by Bob. That is, we will prove that if Bob chooses his vectors such that $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$, then the vector $\boldsymbol{d}[I]$ does not reveal any additional information about Alice's input and Bob's view can be perfectly simulated. Therefore, it suffices to add a mechanism that forces Bob to employ the OT phase with a vector I for which $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$.

4.2 Security against a malicious Alice?

We move on and consider a malicious Alice. Clearly, even if Alice deviates from the protocol and does not compute the vector $\boldsymbol{d}$ properly, her view is still simulatable since all that she sees is a pseudorandom vector. However, such misbehavior may lead Bob to abort and it is not fully clear how to simulate this case. Specifically, let us assume that Alice misbehaves and generates a vector $\boldsymbol{d} \notin \mathsf{colspan}\left(T|\boldsymbol{c}\right)$. The simulator detects this and can send an "abort" to Bob. However, in the real protocol, Bob aborts only if his I-partial view is inconsistent, namely, if $\boldsymbol{d}[I] \notin \mathsf{colspan}\left((T|\boldsymbol{c})[I]\right)$ where $I = I(\boldsymbol{e})$. To make the problem concrete, consider a malicious Alice that honestly computes $\boldsymbol{d}$ and then adds noise to the first coordinate of $\boldsymbol{d}$. In this case, the natural simulator sends an abort, but in the real protocol, Bob aborts only if the first coordinate is in $I(\boldsymbol{e})$ which happens with constant probability $1 - \mu$.

We suggest solving this problem by using a different simulation strategy. Namely, we let the simulator sample a random subset $I' \leftarrow \mathsf{BER}^m(1-\mu)$ and abort if $\boldsymbol{d}[I'] \notin \mathsf{colspan}\left((T|\boldsymbol{c})[I']\right)$. The soundness of this simulation can be reduced to a new, arguably, natural intractability assumption. Since this part will not be used in our subsequent protocols we move it to Appendix A.

5 RVOLE Protocol against Actively-Corrupted Receiver

In this section, we construct a protocol for RVOLE, which is actively secure against Bob and passively secure against Alice. The protocol is based on the ADINZ protocol. We will later use this protocol as a building block of an actively secure VOLE protocol of width w over the field family $\mathbb{F}$. Our protocol relies on *CDS for span membership*. Formally, let $f_{T,\boldsymbol{c}} : \{0,1\}^m \to \{0,1\}$ be a predicate that receives a vector $I \in \{0,1\}^m$ and accepts iff $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$. We will need a CDS $(\mathsf{Enc}_0, \mathsf{Enc}_1, \ldots, \mathsf{Enc}_m)$ for the predicate $f_{T,\boldsymbol{c}}$ (where T and $\boldsymbol{c}$ are viewed as public parameters). We let $\mathsf{Enc}_i(I_i, S; R)$ denote the encoding function of the ith server where $I_i \in \{0,1\}$ is the ith entry of the vector I and the input of the ith server, $S \in \mathbb{F}$ is a secret and R is a vector of common random elements. Furthermore, we assume that one can compute the "global" encoding function $\mathsf{Enc}(I, S; R) = (\mathsf{Enc}_0(S; R), \mathsf{Enc}_1(I_1, S; R), \ldots, \mathsf{Enc}_k(I_m, S; R))$ and the decoding function Dec on I and $m+1$ messages by making only $O(m)$ arithmetic operations over $\mathbb{F}$ (assuming $\mathbb{F}$ is given). Construction of such a CDS with unconditional information-theoretic security appears in Section 8.

Protocol 2 (modified RVOLE protocol). *To initialize the protocol Bob samples the matrix* $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$ *and sends it to Alice.*

1. ***Bob:*** *Given an input* $\boldsymbol{a} \in \mathbb{F}_k^w$*, Bob samples vectors* $\boldsymbol{r} \leftarrow \mathbb{F}_k^k$ *and* $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$*, sets* $I = I(\boldsymbol{e})$ *and sends the vector:* $\boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}) + \boldsymbol{e}$ *to Alice.*
2. ***Alice:*** *Given the inputs* $\boldsymbol{b} \in \mathbb{F}_k^w$*,* $x \in \mathbb{F}_k$*, and Bob's message* $\boldsymbol{c} \in \mathbb{F}_k^m$*, samples a random vector:* $\boldsymbol{r'} \leftarrow \mathbb{F}_k^k$ *and a field element* $x' \leftarrow \mathbb{F}$ *and computes the vector* $\boldsymbol{d} = x'\boldsymbol{c} + E_{\boldsymbol{r'}}(\boldsymbol{b})$*.*
3. ***Alice:*** *Samples randomness* R *for the span membership CDS and sets the secret* $\Delta = x - x'$*, and for each* $i \in [m]$ *Alice computes two possible CDS messages* $z_{i,0} = \mathsf{Enc}_i(0, \Delta; R)$ *and* $z_{i,1} = \mathsf{Enc}_i(1, \Delta; R)$*. In addition, Alice computes the CDS offline message by* $z_0 = \mathsf{Enc}_0(\Delta; R)$ *and sends* z_0 *to Bob.*
4. ***Alice and Bob:*** *Invoke* m*-batch OT where the* i*th entry of Alice is the pair*

$$(z_{i,1}, d_i) \qquad \text{and} \qquad z_{i,0}$$

   and Bob uses the vector I *as its selection vector.*
5. ***Bob:***
   - *Collects all the* z*-part of the OT messages into a vector* $\boldsymbol{z} = (z_0, (z_{i,I_i})_{i\in[m]})$*, and applies the CDS decoder to recover the CDS secret* $\Delta = \mathsf{Dec}(I, \boldsymbol{z})$*. If decoding fails Bob aborts.*
   - *If* $\boldsymbol{d}[I]$ *is not in* $\mathsf{colspan}\left((T|\boldsymbol{c})[I]\right)$*, Bob aborts. Otherwise, Bob employs the decoding-under-erasures property of the gadget* E *(property 3), computes the vector* $\boldsymbol{v'}$ *(supposedly* $x'\boldsymbol{a} + \boldsymbol{b}$*), shifts it by* $\Delta\boldsymbol{a}$ *and outputs the result* $\boldsymbol{v} = \boldsymbol{v'} + \Delta\boldsymbol{a}$ *(supposedly,* $x\boldsymbol{a} + \boldsymbol{b}$*).*

The original ADINZ protocol is obtained by removing the blue parts and setting $x' = x$ and outputting $\boldsymbol{v'}$. As always, we assume the existence of an ideal m-batch OT channel. Through the analysis of Protocol 2, we assume that all the protocol's length parameters: m, w, v and u are polynomial functions of the security parameter k.

Lemma 5.1. *Under Assumption 4.1, the Protocol 2 realizes the RVOLE functionality of width* w *over* $\mathbb{F}$ *in the OT-hybrid model with arithmetic complexity of* $O(w)$ *(ignoring the initialization cost) and with the following guarantees:*

1. *Computational security against a passive adversary that corrupts Alice.*
2. *Computational privacy against an active adversary that corrupts Alice.*
3. *Perfect security against a passive adversary that corrupts Bob.*
4. *Perfect security against an active adversary that corrupts Bob and deviates from the protocol.*

Some comments are in place:

1. (Computationally-unbounded Bob) The information-theoretic security against Bob holds even if the ideal OT channel is replaced with an OT protocol that provides statistical privacy for the sender (e.g., [NP01, AIR01]).

2. (Full security against active Alice) We do not know if Protocol 2 provides full security against an actively corrupt Alice and leave this as an open question. It seems reasonable to assume that the protocol achieves full security. Under this assumption, one can plug Protocol 2 to the standard RVOLE-to-VOLE transformation (Remark 3) and derive an actively-secure VOLE protocol. We refer to this protocol as *VOLE1*.

3. (Concrete communication complexity of CDS) Our concrete CDS (Section 8) communicates $n+1 = k+w+1$ field elements in the offline message z_0 and leaves the 1-messages $z_{i,1}$ empty. Accordingly, each of the OT messages is just a single field element. Moreover, by resorting to computationally-private CDS (and exploiting PRGs), we can use an economic variant of the CDS in which each of the 0-messages, $z_1, \ldots, z_m$, is of length k independently of the field size (see Section 8). As a result, the total communication complexity of the OT messages can be reduced to $m \log |\mathbb{F}_k| + m \cdot k$. Furthermore, this can be done while keeping the computational complexity linear.[17]

4. (On the achievable rate) Based on the aforementioned optimized CDS, Protocol 2 communicates m field elements from Bob to Alice, $n+1 = k+w$ field elements from Alice to Bob in the offline CDS message, and m field elements plus $O(mk)$ bits over the OT-channel. Overall, the number of field elements that are communicated is $2m+n+1 = 2(u+v)+w+k+1 = (2v+w)(1+o(1))$ where the last equality holds since $k = o(w)$ and $u = o(v)$. Recall that v is the length of the code produced by Ecc, which needs to be at least approximately $\frac{1}{1-\mu}w$ to allow successful decoding of w field elements values from a noisy codeword with a fraction of μ random erasures. Therefore, the communication rate of the protocol, measured as the length of the protocol's output w, divided by the communication complexity, approaches to:

$$\frac{w}{2v+w} = \frac{1-\mu}{3-\mu}.$$

   If Assumption 4.1 holds for any constant error rate $\mu > 0$ then we can obtain a rate approaching $\frac{1}{3} - \varepsilon$ for any constant $\varepsilon > 0$. Furthermore, by choosing a non-constant erasure fraction of $\mu = \frac{1}{f(k)}$ for $f(k)$ that tends to infinity with k (for example $f(k) = \frac{1}{\log k}$) we get an asymptotic rate of $1/3$, namely, in order to realize an RVOLE functionally of size w by our protocol $3w$ fields elements should be communicated (where w and k tend to infinity).[18] The reduction to VOLE increases the communication by w additional field elements and so the rate of the VOLE1 protocol is $\frac{1-\mu}{2(2-\mu)}$ which approaches to $1/4$ for a small noise rate. Recall that the communication rate of the passively-secure ADINZ VOLE protocol approaches $1/3$.

We proceed with a partial proof of Lemma 5.1 for the first 3 items and postpone the (interesting) case of actively corrupt Bob to Section 5.1.

Proof of Lemma 5.1, items 1–3. The ADINZ protocol has a time complexity of $O(m)$ (follows by properties 1 and 3 of the protocol's encoder). Our CDS global message is generated and decoded in time of $O(m)$ and hence overall our modified protocol runs in a time of $O(m) = O(w)$. We analyze the passive security of the modified protocol, starting with perfect correctness. By the correctness

[17] Recall that the computational complexity and communication complexity of batch-OT is measured as the total bit-length of the sent messages; see Remark 2 for a justification for this convention.

[18] In the context of binary codes, LPN-style assumptions with sub-constant μ are quite standard.

of the ADINZ protocol, the vector $\boldsymbol{v}'$ computed by Bob equals, except with negligible probability, to $x'\boldsymbol{a}+\boldsymbol{b}$. Bob recovers the CDS secret Δ correctly (recovery succeeds since $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$), and so $\boldsymbol{v} = (x'+\Delta)\boldsymbol{a}+\boldsymbol{b}$ which, by the definition of Δ and the perfect correctness of the CDS, equals to $x\boldsymbol{a}+\boldsymbol{b}$. To simulate a *semi-honest* Bob's view, given an input $\boldsymbol{a}$ and an output $\boldsymbol{v}$: (1) we sample $\Delta \leftarrow \mathbb{F}_k$ and compute $\boldsymbol{v}' = \boldsymbol{v} - \Delta\boldsymbol{a}$; (2) We invoke the ADINZ perfect simulator over input $\boldsymbol{a}$ and output $\boldsymbol{v}'$ and sample $\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{d}[I(\boldsymbol{e})]$; (3) We honestly generate the CDS messages with fresh randomness and secret Δ (mimicking the behavior of Alice). It is not hard to verify that the simulation is perfect. Finally, Alice's view consists of M and $\boldsymbol{c}$ and her random tape (regardless of her behavior in the protocol), which can be simulated by sampling M properly and by sampling a random vector $\boldsymbol{c}$. Privacy follows from Assumption 4.1. □

5.1 Proof of security against a malicious Bob

We present a simulator that perfectly emulates the view of a malicious Bob $\mathrm{B}^*(\boldsymbol{a})$. The simulator makes a black-box use of B^* in a straight-line way (without rewinding).

Simulator 1 (Bob's Simulator). $\textsc{Sim}_{\mathrm{B}^*}(\boldsymbol{a})$

1. *Apply* B^* *on* $\boldsymbol{a}$*, get a matrix* M*, a vector* $\boldsymbol{c} \in \mathbb{F}_k^m$ *and extract the vector of OT choices* $I \in \left\{0,1\right\}^m$ *of* B^**. Extract an effective input* $\boldsymbol{a}'$ *as follows:*
   - *If* $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$*, extract Bob's "effective input"* $\boldsymbol{a}' \in \mathbb{F}_k^w$ *by finding a solution to the linear system:*
   $$\boldsymbol{c}[I] = T[I] \cdot \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{a}' \end{pmatrix}$$
   - *Else, set* $\boldsymbol{a}' := 0^w$*.*

   Call the ideal functionality with $\boldsymbol{a}'$ *and get* $\boldsymbol{v}$*.*
2. *Fix some arbitrary* $\hat{x} \in \mathbb{F}_k$ *and set* $\hat{\boldsymbol{b}} := \boldsymbol{v} - \hat{x}\boldsymbol{a}'$ *and compute the vector* $\boldsymbol{d}$ *and the CDS message vector* $\boldsymbol{z}$ *exactly as in the protocol with respect to the inputs* $(\hat{x}, \hat{\boldsymbol{b}})$*. That is, let*
   $$\boldsymbol{r}'_S \leftarrow \mathbb{F}_k^k, \quad x'_S \leftarrow \mathbb{F}, \quad \boldsymbol{d} := x'_S\boldsymbol{c} + E_{\boldsymbol{r}'_S}(\hat{\boldsymbol{b}}),$$
   and similarly, sample randomness R *for the span membership CDS, and set*
   $$\Delta := \hat{x} - x'_S, \quad z_0 := \mathsf{Enc}_0(\Delta; R), \quad z_{i,b} = \mathsf{Enc}_i(b, \Delta; R), \quad \forall i \in [m], b \in \{0,1\}\,.$$
   Give B^* *the tuple* $\left(z_0, \{(z_{i,1}, d_i)\}_{i\in[m], I_i=1}, \left\{z_{i,0}\right\}_{i\in[m], I_i=0}\right)$ *and halt with the output of* B^**.*

Since the functionality does not send any output to Alice it suffices to analyze the view of Bob.

Lemma 5.2. *For every* B^**, security parameter* k*, and inputs* $\boldsymbol{a}, \boldsymbol{b}, x \in \mathbb{F}_k^w \times \mathbb{F}_k^w \times \mathbb{F}_k$ *the simulator* $\textsc{Sim}_{\mathrm{B}^*}(\boldsymbol{a})$ *perfectly samples the view of* B^* *in the ideal execution.*

Proof. Fix the randomness of B^*. To prove the claim it suffices to show that the tuple
$$\left(z_0, \{(z_i, d_i)\}_{i\in[m], I_i=1}, \{z_i\}_{i\in[m], I_i=0}\right)$$

is identically distributed in both experiments, the real protocol, and the simulation. Through we fix some arbitrary value for $\hat{x}$ in the simulation.

(1) Let us first condition on the event that $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$. Let x'_S denote the simulated copy of x' and let x'_R denote the copy of x' in the real execution. Recall that both are uniformly distributed and so we can condition on $x'_R = x - \hat{x} + x'_S$. In this case, $\hat{x} - x'_S = x - x'_R$ and so the shift Δ takes the same value both in the simulation and in the real experiment. Accordingly, the CDS part in both experiments is distributed identically. (In fact, *all* the CDS messages, including the ones that are not delivered to B^*, are distributed identically.) We move on to analyze the distribution of $\boldsymbol{d}[I]$. By assumption, we can write

$$\boldsymbol{c}[I] = T[I] \cdot \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{a}' \end{pmatrix} = E_{\boldsymbol{r}}(\boldsymbol{a}')[I]$$

where $\boldsymbol{a}' \in \mathbb{F}_k^w$ is the vector that was extracted by the simulator and $\boldsymbol{r} \in \mathbb{F}_k^k$ is some vector. Let $\boldsymbol{v} = x\boldsymbol{a}' + \boldsymbol{b}$ be the output that the ideal functionality returns to the simulator, and note that the simulated vector $\hat{\boldsymbol{b}} = \boldsymbol{v} - \hat{x}\boldsymbol{a}' = (x - \hat{x})\boldsymbol{a}' + \boldsymbol{b}$. Then, by definition, the simulated $\boldsymbol{d}[I]$ can be written as

$$\begin{aligned}
\boldsymbol{d}[I] &= x'_S E_{\boldsymbol{r}}(\boldsymbol{a}')[I] + E_{\boldsymbol{r}'_S}(\hat{\boldsymbol{b}})[I] \\
&= x'_S E_{\boldsymbol{r}}(\boldsymbol{a}')[I] + E_{\boldsymbol{r}'_S}((x - \hat{x})\boldsymbol{a}' + \boldsymbol{b})[I] \\
&= E_{x'_S\boldsymbol{r} + \boldsymbol{r}'_S}((x'_S + x - \hat{x})\boldsymbol{a}' + \boldsymbol{b})[I] \\
&= E_{x'_S\boldsymbol{r} + \boldsymbol{r}'_S}(x'_R\boldsymbol{a}' + \boldsymbol{b})[I].
\end{aligned}$$

Since $x'_S\boldsymbol{r} + \boldsymbol{r}'_S$ is distributed uniformly (due to the uniform choice of $\boldsymbol{r}'_S$) the resulting distribution is identical to the distribution in the real experiment.

(2) Now let us consider the event where $\boldsymbol{c}[I] \notin \mathsf{colspan}\left(T[I]\right)$. In this case, $\boldsymbol{a}' = 0^w$ and so the ideal functionality returns $\boldsymbol{v} = x\boldsymbol{a}' + \boldsymbol{b} = \boldsymbol{b}$ and $\hat{\boldsymbol{b}} = \boldsymbol{v} - \hat{x}\boldsymbol{a}' = \boldsymbol{b}$. Conditioning on $x'_R = x'_S$, we conclude that the whole vector $\boldsymbol{d}$ is distributed identically in both experiments. On the other hand, since the input I does not satisfy the CDS condition, the corresponding CDS messages in the simulated experiment (where the secret is $\hat{x} - x'_S$) are distributed identically to the CDS messages in the real execution (where the secret is $x - x'_S$). □

6 Actively-Secure VOLE under Correlated Noisy-Codewords

In this section, we realize the VOLE functionality directly while achieving active security against both Alice and Bob. For this, we introduce an additional, new, "Correlated Noisy-Codeword" intractability assumption. We will also have to slightly modify the parameters of the ADINZ encoding matrix. Recall that the ADINZ encoding matrix T is defined as follows:

$$T = \left(\begin{array}{c|c} & \\ M_{m\times k} & \begin{array}{c} \mathbf{0}_{u\times w} \\ \\ \mathsf{Ecc}_{v\times w} \end{array} \\ & \end{array} \right),$$

where $m = \Omega(k^3)$, $u = \Omega(k \log^2 k)$ and $v = O(m)$. For technical reasons we will need to slightly strengthen the linear-independence requirements of the matrix T as follows. Except with negligible

probability over the choice of M and Ecc it must hold that: (a) If we sample a random subset of the first u rows of M by taking each row independently with probability $1/\log^{1.5} k$ then the resulting matrix has full rank (all the columns are linearly independent); (b) The error-correcting code Ecc can correct up to $O(\log^{1.1} k)$ *errors* and, as before, can recover from say $1.2\mu v$ arbitrary erasures. (The constant 1.2 can be replaced with any constant larger than 1.)

6.1 The Correlated Noisy-Codeword Hardness Assumption

The following intractability assumption intuitively asserts that given a noisy codeword $\boldsymbol{c} = T\boldsymbol{v} + \boldsymbol{e}$ of T, it is hard to efficiently generate a new noisy codeword $\boldsymbol{d} = T\boldsymbol{v}' + \boldsymbol{e}'$ whose noise is non-trivially correlated with $\boldsymbol{e}$ in the following sense. The new noise vector $\boldsymbol{e}'$ agrees with the original noise vector $\boldsymbol{e}$ with respect to the set of non-noisy coordinates $I = I(\boldsymbol{e})$, i.e., $\boldsymbol{d}[I] \in \mathsf{colspan}\left(T[I]\right)$, but $\boldsymbol{e}'$ is "far" from being a scalar multiple of $\boldsymbol{e}$. That is, $\rho\left(\boldsymbol{d}, \mathsf{colspan}\left(T|\boldsymbol{c}\right)\right) = \ell$ where $\rho\left(\boldsymbol{d}, S\right)$ is the minimal Hamming distance between a vector $\boldsymbol{d}$ and a set of vectors $S \subset \mathbb{F}^m$. Observe that such a noisy codeword can be generated by sampling a vector in the column span of $(T|\boldsymbol{c})$ and then modifying ℓ entries with the hope that all these entries fall out of the set of clean coordinates I. Such an attack succeeds with probability μ^ℓ, the following assumption states that this is essentially the best that one can hope for up to polynomial speed-ups.[19]

Assumption 6.1 (Correlated noisy codeword)**.** *For a distribution $\mathcal{T}$ over matrices in $\mathbb{F}_k^{m\times n}$ where $m(k), n(k)$ are some polynomials in the security parameter k, and for a constant noise rate of $\mu < 1/2$, the* Correlated Noisy-Codeword *assumption asserts that for every efficient adversary A^* there exists some negligible $\varepsilon(k)$ and constant C such that for every integer $\ell \leq m$:*

$$\Pr_{\boldsymbol{d}\leftarrow \mathrm{A}^*(\boldsymbol{c})}\left[\boldsymbol{d}[I] \in \mathsf{colspan}\left(T[I]\right) \quad \textit{and} \quad \rho\left(\boldsymbol{d}, \mathsf{colspan}\left(T|\boldsymbol{c}\right)\right) = \ell\right] \leq \exp(-C\ell) + \varepsilon(k)$$

where $T \leftarrow \mathcal{T}$ and $\boldsymbol{c} = T\boldsymbol{v} + \boldsymbol{e}$ for $\boldsymbol{v} \leftarrow \mathbb{F}_k^n$, $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$ and $I = I(\boldsymbol{e})$.

For our purposes, it suffices to assume the "super-logarithmic version of Assumption 6.1" that asserts that for every super-logarithmic function $\ell(k) = \omega(\log k)$ the success probability in the above game is negligible in k. (Note that this variant follows from the above formulation.) We conjecture that every matrix distribution whose noisy codewords are pseudorandom also satisfies this assumption, and provide some evidence for this in Section B. From now, we will always use the super-logarithmic version of the assumption with respect to the distribution $\mathcal{T}$ that corresponds to the ADINZ encoding matrix.

We will make use of the following simple observation.

Lemma 6.2. *There exists a probabilistic polynomial-time algorithm* G *that given the matrix T and the vectors $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{F}_k^m$ outputs a vector $\boldsymbol{u} \in \mathbb{F}_k^{n+1}$ with the following guarantee. Except with negligible probability in k over the choice of $(T, \boldsymbol{c})$ (which are distributed as in Assumption 6.1) and the randomness of* G*, if $\boldsymbol{d}$ is ℓ-close to $\mathsf{colspan}\left(T|\boldsymbol{c}\right)$ for $\ell = O(\log^{1.1} k)$ then the algorithm outputs $\boldsymbol{u}$ such that $(T|\boldsymbol{c}) \cdot \boldsymbol{u}$ is ℓ-close to $\boldsymbol{d}$.*

[19] The concrete formulation that is taken here is chosen for the sake of simplicity. More refined and conservative versions (e.g., that assume better speed-ups and consider sub-constant noise regimes) can be adopted as well.

Proof. The algorithm $\mathrm{G}(T, \boldsymbol{c}, \boldsymbol{d})$ decomposes $(T, \boldsymbol{c}, \boldsymbol{d})$ into a top part $(T_1, \boldsymbol{c}_1, \boldsymbol{d}_1)$ that contains the first u rows (resp., elements) and a bottom part $(T_2, \boldsymbol{c}_2, \boldsymbol{d}_2)$ that contains the last v elements. Similarly decompose M to a top part M_1 and a bottom part M_2 and recall that $T_1 = (M_1|0_{u\times w})$ and $T_2 = (M_2|\mathsf{Ecc})$. The algorithm repeatedly applies the following procedure as long as the number of failed iterations does not exceed k:

1. Sample a random vector $I' \leftarrow \mathsf{BER}^u(\mu')$ where $\mu' = 1/\log^{1.5} k$ and find a vector $(\boldsymbol{r}', x') \in \mathbb{F}_k^{k+1}$ that satisfies the linear system $M_1[I']\boldsymbol{r}' + x'\boldsymbol{c}_1[I'] = \boldsymbol{d}_1[I']$. If there is no solution to the system, this iteration is aborted with failure.
2. Let $\boldsymbol{a}_2 = M_2\boldsymbol{r}' + x'\boldsymbol{c}_2 \in \mathbb{F}_k^v$. Use the efficient error-correction to find a vector $\boldsymbol{b}' \in \mathbb{F}_k^w$ such that $\mathsf{Ecc}(\boldsymbol{b}')$ is ℓ-close to the vector $\boldsymbol{d}_2 - \boldsymbol{a}_2$. Aggregate the solution into the vector $\boldsymbol{u} = (\boldsymbol{r}', \boldsymbol{b}', x')$. If no solution is found, this iteration is aborted with failure.

Analysis. Let $\boldsymbol{d}^* = (T|\boldsymbol{c}) \cdot \boldsymbol{u}^*$ be a closest vector to $\boldsymbol{d}$ in $\mathsf{colspan}\left(T|\boldsymbol{c}\right)$ and let $\boldsymbol{u}^* = (\boldsymbol{r}^*, \boldsymbol{b}^*, x^*)$. Assume that $\boldsymbol{d}^*$ and $\boldsymbol{d}$ are ℓ-close, and denote by $L \subset [m]$ the subset of coordinates on which $\boldsymbol{d}^*$ and $\boldsymbol{d}$ disagree. For the sake of analysis, we define the following "bad" events.

For every iteration $i \in [k]$, denote by A_i the event that the ith subset I' hits some location in L, let B_i denote the event that $\boldsymbol{e}[I']$ is in the image of $M_1[I']$, let C_i denote the event that $M_1[I']$ does not have full rank, and let D_i denote the event that decoding fails at the second step (i.e., that the given vector is ℓ-close to a codeword but decoding fails). First observe that if in some iteration i, neither A_i nor B_i nor C_i nor D_i occur then the corresponding solutions $(\boldsymbol{r}', x') \in \mathbb{F}_k^{k+1}$ form unique solutions to the linear system $M_1[I']\boldsymbol{r}' + x'\boldsymbol{c}_1[I'] = \boldsymbol{d}^*[I']$ since $(M_1|\boldsymbol{c}_1)[I']$ has a full rank. Hence, $(\boldsymbol{r}', x') = (\boldsymbol{r}^*, x^*)$, which means that $\mathsf{Ecc}(\boldsymbol{b}^*)$ is ℓ-close to the vector $\boldsymbol{d}_2 - (M_2\boldsymbol{r}^* + x^*\boldsymbol{c}_2)$, and so the decoding recovers $\boldsymbol{b}^*$, as required.

Let us analyze the success probability. Fix i. First, observe that by properties (a) and (b) (as defined at the beginning of the section), C_i and D_i happens with negligible probability. To upper-bound B_i observe that by multiplicative Chernoff bound except with negligible probability $\exp(-\Omega(u\mu\mu')) = \exp(-\Omega(k))$, the support of $\boldsymbol{e}[I']$ is of size at least $u\mu\mu'/2 \geq 2k$, where the inequality holds since μ is assumed to be constant and u is super-linear in k.[20] Conditioned on this event, the probability that $\boldsymbol{e}[I']$ falls in the image of $M_1[I']$ is at most $|\mathbb{F}_k|^{k-2k}$ which is negligible. By a union bound over all iterations, we conclude that the probability that B_i or C_i or D_i happens for some i, is negligible. We move on to analyze A_i. This event happens with probability at most $|L|\mu' \leq 1/\log^{0.4} k = o(1)$. Here the probability is taken over the ith choice of I' and so the events $A_1, \ldots, A_k$ are statistically independent. Therefore, the probability that all A_i's fail is negligible in k. The lemma follows. □

6.2 The VOLE2 protocol

We present our VOLE protocol and prove security under Assumption 6.1. The protocol employs an ideal-commitment functionality, aka *commitment channel*, which is a 2-phase ideal functionality of the following form. In the commit phase, the functionality takes an input x from a sender (e.g., a field element), and delivers a commit message to the receiver. At a later phase, the sender can de-commit by sending an "open" message to the functionality which delivers to the receiver the committed message x.

[20] Indeed, one can take μ to be slightly sub-constant as well.

Remark 6 (Realizing ideal commitment)**.** *It is well known that an ideal commitment channel can be constructed based on OT-channel [Cré87, CK88]. For our purpose, the following simple variant suffices. To commit a single field element $x \in \mathbb{F}_k$ the Sender sends via the ideal-OT channel k pairs $(m_{i,0}, m_{i,1})_{i\in[k]}$ where $(m_{i,0}, m_{i,1})$ is a pair of random field elements that satisfies $m_{i,0} + m_{i,1} = x$. The Receiver selects a random entry from each pair, i.e., samples, for each i a value $s_i \leftarrow \{0,1\}$ and receives m_{s_i}. To open the commitment the sender reveals all pairs $(m_{i,0}, m_{i,1})$ and x and the receiver accepts if and only if each of these pairs is consistent with x and with the received value in the i-th OT. The communication and computational complexity is k OT-calls and k field additions. The proof of security is standard (see, e.g., [Kil88] for proof of a closely related construction). In the OT-hybrid model, we get perfect security against the receiver and statistical security against the sender. We mention that there are more efficient constructions that achieve a constant rate for long messages, see [GIKW14].*

Protocol 3 (**VOLE2 protocol**)**.** *To initialize the protocol Bob samples the matrix $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$ and sends it to Alice.*

1. ***Alice and Bob:*** *Hold inputs $x \in \mathbb{F}_k$ and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_k^w$ respectively. The parties invoke Protocol 2 for RVOLE where Bob's input is a random vector $\boldsymbol{a}' \leftarrow \mathbb{F}_k^w$, and Alice's input is x and a random vector $\boldsymbol{b}' \leftarrow \mathbb{F}_k^w$. Let x' denote the random field element that is being sampled by Alice in the protocol and let $\Delta_x = x - x'$ denote the secret that Alice delivers via the CDS.*
2. ***Alice:*** *Sends Δ_x over an ideal commitment channel which delivers a "commit" message to Bob.*
3. ***Bob:*** *If Bob aborts during Protocol 2 then he aborts the entire execution. Otherwise, Bob recovers from the protocol the vector $\boldsymbol{v}'$ (supposedly $x'\boldsymbol{a}' + \boldsymbol{b}'$) and the CDS secret Δ_x (supposedly $x - x'$). Bob sends Δ_x to Alice.*
4. ***Alice:*** *Verifies that Bob's message equals to Δ_x, and aborts if the check fails. If the check passes, Alice decommits by sending an "open" message to the commitment channel which delivers the committed value, Δ_x to Bob.*
5. ***Bob:*** *Verifies that the decommitted value equals to Δ_x, and aborts if the check fails. If the check passes, Bob sends to Alice the vectors $\boldsymbol{v} = \boldsymbol{v}' + \Delta_x\boldsymbol{a}' + \boldsymbol{b}$ (supposedly, $x\boldsymbol{a}' + \boldsymbol{b}' + \boldsymbol{b}$) and $\boldsymbol{\Delta_a} = \boldsymbol{a} - \boldsymbol{a}'$.*
6. ***Alice:*** *Computes the vector $\boldsymbol{w} = x\boldsymbol{\Delta_a} + \boldsymbol{v} - \boldsymbol{b}'$ (supposedly, $x\boldsymbol{a} + \boldsymbol{b}$) and outputs the result.*

In section 6.3 we prove that the protocol is computationally secure against an actively corrupt Alice, and in Section 6.4 we prove that the protocol is statistically secure against an actively corrupt Bob. By replacing the commitment with k calls to OT(Remark 6), we derive the following lemma.

Lemma 6.3. *Suppose that Assumptions 4.1 and 6.1 hold. Then protocol 3 for honest parties realizes the VOLE functionality of width w over $\mathbb{F}$ with arithmetic complexity of $O(w)$ (ignoring the initialization cost) in the OT-hybrid model. The protocol is statistically-secure against an active sender Bob with negligible deviation error and computationally secure against an active receiver Alice.*

Some comments are in place:

1. (Unbounded sender) Here too, the protocol achieves information-theoretic security against the sender even if the ideal channels are replaced by an OT protocol that provides statistical

privacy for the sender and by a commitment scheme that is statistically hiding. (The latter reduces to the former by using the OT-to-Commitment transformationfrom Remark 6).

2. (Working over small fields) The statistical error is exponentially-small in the bit-length of the field element Δ_x. (This essentially corresponds to the case where Bob guesses the value of Δ_x despite playing dishonestly in the RVOLE protocol in a way that keeps the CDS secret hidden). Thus, when the field is small the error is only $1/|\mathbb{F}_k|$. Nevertheless, even when the field is small, one can easily get a negligible error at a minor cost by randomly padding the element Δ_x to length k.

3. (On the achievable rate of Protocol 3) The communication of Protocol 3 (VOLE2) consists of $(2v + w)(1 + o(1))$ field elements in Step 1 (when invoking the RVOLE Protocol), $2k$ field elements to commit (via k OT calls) and to de-commit, and additional $2w+1$ elements. Since $k = o(w)$, the total communication complexity is $(2v + 3w)(1 + o(1))$. Recall that v is the length of the code produced by Ecc, which needs to be at least approximately $\frac{1}{1-\mu}w$ to allow successful decoding of w field elements values from a noisy codeword with fraction of μ random erasures. Therefore, the communication rate of the protocol, measured as the length of the protocol's output w, divided by the communication complexity, approaches to

$$\frac{w}{2v + 3w} = \frac{1 - \mu}{5 - 3\mu}.$$

   If Assumption 4.1 holds for any constant error rate $\mu > 0$ then the rate of RVOLE2 approaches to $\frac{1}{5} - \varepsilon$ for any constant $\varepsilon > 0$.

4. (Comparison to VOLE1) In terms of communication we pay an amortized cost of an extra field element per each VOLE entry compared to VOLE1, which, in turn, pays an extra field element compared to the passively-secure ADINZ protocol. In terms of computation, VOLE2 has a negligible overhead compared to VOLE1 which consists of a single commitment (for the entire VOLE), and, an amortized cost of $1/R$ field multiplication and $2/R$ field additions per VOLE entry where $R = w/v$ is the rate of the error-correcting code.[21]

6.3 Proof of security against a malicious Alice

Let A^* be an adversary that corrupts the VOLE receiver, Alice. We present a simulator that participates in the ideal experiment and computationally emulates the real execution. The simulator makes black-box use of A^* in a straight-line way (without rewinding) and makes a single call to the ideal VOLE functionality that given a field element $g \in \mathbb{F}_k$ returns the vector $g\boldsymbol{a} + \boldsymbol{b}$ where $\boldsymbol{a}, \boldsymbol{b}$ are the inputs of (honest) Bob. In addition, we implicitly assume that the simulator is given a matrix M that is sampled from $\mathcal{M}(1^k, \mathbb{F})$ as an auxiliary input. (This only makes the statement stronger as it guarantees that M can be reused - See Remark 5.)

[21] Recall that VOLE1 is obtained by combining the RVOLE-to-VOLE transformation from Remark 3 with Protocol 2 for RVOLE. Accordingly, the latter protocol achieves provable active security against the Sender, provable passive security against the Receiver, and heuristic active security against the Receiver.

Simulator 2 (**Alice's Simulator**). $\text{SIM}_{\text{A}^*}(x)$

1. *Sample a uniform vector* $\boldsymbol{a}' \leftarrow \mathbb{F}_k^w$*, compute the noisy codeword* $\boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}') + \boldsymbol{e}$ *for* $\boldsymbol{r} \leftarrow \mathbb{F}_k^k$ *and* $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$ *and sets* $I = I(\boldsymbol{e})$*.*
2. *Feed* A^* *with the input* x *and the first message* $\boldsymbol{c}$*, and get back the offline CDS message,* z_0*, the OT messages* $\left\{z_{i,0}, (z_{i,1}, d_i)\right\}_{i\in[m]}$ *and a message* Δ_x *that is directed to the ideal commitment channel.*
3. *If* $\boldsymbol{d}[I] \notin \mathsf{colspan}\left(T[I]\right)$*, send to* A^* *an abort message on behalf of Bob and halt with the output of* A^**.*
4. *Recover* A^**'s "effective" CDS secret by applying the CDS decoding function with respect to the vector* I*, i.e., compute* $y \leftarrow \mathsf{Dec}\left(I, z_0, \{z_{i,I_i}\}_{i\in[m]}\right)$*. Send* y *to* A^**.*
5. *If* A^* *doesn't respond with an "open" message, or if* y *is not equal to the committed message* Δ_x*, send to* A^* *an abort message on behalf of Bob and halt with the output of* A^**.*
6. *Use the algorithm* G *described in Lemma 6.2 in order to find the vector* $\boldsymbol{u} = (\boldsymbol{r}', \boldsymbol{b}', x')$ *such that* $(T|\boldsymbol{c})\boldsymbol{u}$ *is within Hamming distance of at most* $\log^{1.1} k$ *from the vector* $\boldsymbol{d}$*. If we fail to find such a vector* $\boldsymbol{u}$*, the simulation halts with a special failure symbol* $\perp$*.*
7. *Set* $x'' = x' + \Delta_x$*, call the ideal functionality with* x'' *and get the vector* $\boldsymbol{v}'$*.*
8. *Sample the vector* $\boldsymbol{\Delta_a} \leftarrow \mathbb{F}_k^w$ *and compute* $\boldsymbol{v} = \boldsymbol{v}' - x''\boldsymbol{\Delta_a} + \boldsymbol{b}'$*, send both to* A^* *and halt with its output.*

We analyze the simulator.

Lemma 6.4. *For every efficient* A^* *and ensemble of inputs* $\left\{(\boldsymbol{a}_k, \boldsymbol{b}_k, x_k) \in \mathbb{F}_k^w \times \mathbb{F}_k^w \times \mathbb{F}_k\right\}_{k\in\mathbb{N}}$*, the simulator* $\text{SIM}_{\text{A}^*}(x_k)$ *computationally emulates* A^**.*

Proof. Fix the randomness of A^* and ensemble of inputs $\left\{\boldsymbol{a}_k, \boldsymbol{b}_k, x_k \in \mathbb{F}_k^w \times \mathbb{F}_k^w \times \mathbb{F}_k\right\}_{k\in\mathbb{N}}$. Note that up to Step 5 (including) the simulation perfectly mimics the real experiment, and so if an abort occurs in one of these steps we have a perfect simulation. Let us, therefore, condition on the event that both the real and ideal executions do not abort during the first five steps. Under this conditioning, Bob does not abort (and has no other output), and so it suffices to show that the output of the simulator is computationally indistinguishable from the output of A^* in a real execution of the protocol where Bob's inputs are $\boldsymbol{a}_k, \boldsymbol{b}_k$. We prove this by a hybrid argument.

Hybrid H**.** Let H be the experiment in which the simulator SIM_{A^*} computes the vector $\boldsymbol{\Delta_a}$ in step 8 by $\boldsymbol{\Delta_a} = \boldsymbol{a}_k - \boldsymbol{a}'$ and other than that operates exactly as before.

Claim 6.5. *Except with negligible probability, the view of* A^* *that is generated by* H *is distributed the same way as the view in the real experiment.*

Proof of Claim. By Assumption 6.1, conditioned on $\boldsymbol{d}[I] \in \mathsf{colspan}\left(T[I]\right)$, it must hold, except with negligible probability, that $\boldsymbol{d}$ is ℓ-close to $\mathsf{colspan}\left(T|\boldsymbol{c}\right)$ for some $\ell \leq \log^{1.1} k$. Therefore, by Lemma 6.2, the algorithm G succeeds, except with negligible probability. From now on, we condition on this event.

Observe that the view of A^* in H,

$$\left(M, \boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}') + \boldsymbol{e}, \boldsymbol{\Delta_a} = \boldsymbol{a}_k - \boldsymbol{a}', \boldsymbol{v}_H\right),$$

and the view of A^* in the real experiment,

$$\left(M, \boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}') + \boldsymbol{e}, \boldsymbol{\Delta}_{\boldsymbol{a}} = \boldsymbol{a}_k - \boldsymbol{a}', \boldsymbol{v}_R\right),$$

are identical except for their last entry. Fix some $M, \boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}') + \boldsymbol{e}$ and $\boldsymbol{d} \leftarrow \mathrm{A}^*(\boldsymbol{c})$ such that $\boldsymbol{d}[I] \in \mathsf{colspan}\left(T[I]\right)$. Observe that

$$\boldsymbol{v}_H = \boldsymbol{v}'_H - x''\boldsymbol{\Delta}_{\boldsymbol{a}} + \boldsymbol{b}' = (x' + \Delta_x)\boldsymbol{a}' + \boldsymbol{b}' + \boldsymbol{b}_k,$$

where the equality follows by plugging-in $\boldsymbol{v}'_H = x''\boldsymbol{a}_k + \boldsymbol{b}_k$, $x'' = (x' + \Delta_x)$ and $\boldsymbol{a}' = (\boldsymbol{a}_k - \boldsymbol{\Delta}_{\boldsymbol{a}})$. On the other hand, in the real experiment

$$\boldsymbol{v}_R = \boldsymbol{v}' + \Delta_x\boldsymbol{a}' + \boldsymbol{b}_k$$

where $\boldsymbol{v}'$ is derived as the output of the RVOLE sub-protocol. So in order to show that $\boldsymbol{v}_R = \boldsymbol{v}_H$, it suffices to show that $\boldsymbol{v}' = x'\boldsymbol{a}' + \boldsymbol{b}'$. Recall that $\boldsymbol{v}'$ is the solution of the system:

$$\boldsymbol{d}[I] = T[I]\begin{pmatrix}\boldsymbol{s}\\ \boldsymbol{v}'\end{pmatrix}$$

for some $\boldsymbol{s}$ and $I = I(\boldsymbol{e})$. Also, by definition, $(\boldsymbol{r}', \boldsymbol{b}', x') \leftarrow \mathrm{G}(T, \boldsymbol{c}, \boldsymbol{d})$ are solutions to the system:

$$\boldsymbol{d}[I^*] = T[I^*]\begin{pmatrix}\boldsymbol{r}'\\ \boldsymbol{b}'\end{pmatrix} + x'\boldsymbol{c}[I^*] \tag{2}$$

for some "large" set $I^* \in \left\{0,1\right\}^m$ of size at least $m - \log^{1.1} k$. Let $I^*_{\boldsymbol{e}} = I \cap I^*$ where, by abuse of notation, we identify a binary vector $J \in \{0,1\}^m$ with the m-subset $\left\{j : J_j = 1\right\}$. Recall that by the definition of $\boldsymbol{c}$, it holds that $\boldsymbol{c}[I^*_{\boldsymbol{e}}] = T[I^*_{\boldsymbol{e}}]\begin{pmatrix}\boldsymbol{r}\\ \boldsymbol{a}'\end{pmatrix}$. By plugging this into (2), we get that

$$\boldsymbol{d}[I^*_{\boldsymbol{e}}] = T[I^*_{\boldsymbol{e}}]\begin{pmatrix}\boldsymbol{r}' + x'\boldsymbol{r}\\ \boldsymbol{b}' + x'\boldsymbol{a}'\end{pmatrix} = T[I^*_{\boldsymbol{e}}]\begin{pmatrix}\boldsymbol{s}\\ \boldsymbol{v}'\end{pmatrix}.$$

To complete the argument, it suffices to show that, except with negligible probability, $T[I^*_{\boldsymbol{e}}]$ has full rank. Let us decompose T and M into a top part T_1 and M_1 that contains the first u rows, and into a bottom part T_2 and M_2 that contains the last v rows. Similarly, we decompose $I^*_{\boldsymbol{e}}$ to $I^*_{\boldsymbol{e},1} := I^*_{\boldsymbol{e}} \cap \{1, \ldots, u\}$ and $I^*_{\boldsymbol{e},2} := I^*_{\boldsymbol{e}} \cap \{u+1, \ldots, m\}$. Recall that $T_1 = (M_1|0_{u\times w})$ and $T_2 = (M_2|\mathsf{Ecc})$. Thus, to prove that $T[I^*_{\boldsymbol{e}}]$ has full rank it suffices to show that (1) $M_1[I^*_{\boldsymbol{e},1}]$ has full rank and (2) $\mathsf{Ecc}[I^*_{\boldsymbol{e},2}]$ has full rank.

Let us start with (2). By a Chernoff bound, except for negligible probability, the support of $I \cap \{u+1, \ldots, m\}$ is at least $(1 - 1.1\mu)(m-u) = (1-1.1\mu)v$, and so the support of $I^*_{\boldsymbol{e},2}$ is of size at least $(1-1.1\mu)v - \log^{1.1} k$ which is at least $(1-1.2\mu)v$ for sufficiently large k's. Thus $\mathsf{Ecc}[I^*_{\boldsymbol{e},2}]$ has full rank except with negligible probability due to our assumption on Ecc (property (b) as stated at the beginning of the section).

We move on to analyze (1). We can think about $I^*_{\boldsymbol{e},1}$ as being generated via the following 2-move game: First, a matrix $M \leftarrow \mathcal{M}$ is sampled as well as a random set $I_1(\boldsymbol{e}) \leftarrow \mathsf{BER}^u(1-\mu)$ and then an adversary removes at most $\log^{1.1} k$ elements from $I_1(\boldsymbol{e})$ and generates $I^*_{\boldsymbol{e},1}$ in an attempt to reduce

the rank of $M_1[I^*_{\boldsymbol{e},1}]$ below k. Let us denote by p the winning probability of the adversary. To bound p, consider the following related experiment: Sample $M \leftarrow \mathcal{M}$ and $I \leftarrow \mathsf{BER}^u(1-\mu)$, and then remove each entry of I with probability $\alpha = 1 - \frac{1}{(1-\mu)\log^{1.5} k}$ and denote by I' the resulting set. Let q denote the probability that $M_1[I']$ is not of full rank. On one hand,

$$q \geq p \cdot \alpha^{\log^{1.1} k} \geq p \cdot \left(1 - \frac{1}{(1-\mu)\log^{1.5} k}\right)^{\log^{1.5} k} \geq \Omega(p),$$

where the first inequality follows by considering the event in which the set of randomly removed elements in the second step of the second experiment contains the adversarially removed elements in step (2) of the first experiment. On the other hand, the second experiment is equivalent to sampling a subset from the distribution $\mathsf{BER}^u((1-\mu)(1-\alpha)) = \mathsf{BER}^u(\frac{1}{\log^{1.5} k})$ and so $q = \mathsf{negl}(k)$ by property (a) of the distribution $\mathcal{M}$ (as defined at the beginning of the section). This completes the proof of the claim. □

We complete our proof with the following claim:

Claim 6.6. *The view of* A^* *that is generated by* H *is computationally indistinguishable from the view that is generated by* $\textsc{Sim}_{\mathrm{A}^*}(x_k)$ *in the ideal experiment.*

Proof of Claim. The view of Alice consists of the matrix M the noisy codeword $\boldsymbol{c}$, the shift vector $\boldsymbol{\Delta_a}$, and the vector $\boldsymbol{v}$. In both experiments the entry $\boldsymbol{v}$ is computed by applying the same efficiently computable function to the first 3 entries of the distribution, therefore, we can omit it from the view. It suffices to show that the tuple $(M, \boldsymbol{c}, \boldsymbol{\Delta_a})$ as distributed in H is computationally indistinguishable from the tuple $(M, \boldsymbol{c}, \boldsymbol{\Delta_a})$ as distributed in the simulation. In the hybrid experiment, this tuple is distributed as

$$(M \leftarrow \mathcal{M}(1^k, \mathbb{F}), \boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}') + \boldsymbol{e}, \boldsymbol{\Delta_a} = \boldsymbol{a}_k - \boldsymbol{a}')$$

where $\boldsymbol{a}' \leftarrow \mathbb{F}_k^w$, $\boldsymbol{r} \leftarrow \mathbb{F}_k^k$ and $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$. This tuple is distributed identically to the tuple

$$(M \leftarrow \mathcal{M}(1^k, \mathbb{F}), \boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}') + \boldsymbol{e}, \boldsymbol{\Delta_a} \leftarrow \mathbb{F}_k^w)$$

where $\boldsymbol{a}' = \boldsymbol{a}_k - \boldsymbol{\Delta_a}$ and the random variables $\boldsymbol{r}$ and $\boldsymbol{e}$ are distributed as before. By the pseudorandomness property of M, it follows that the above tuple is computationally indistinguishable from the tuple

$$(M \leftarrow \mathcal{M}(1^k, \mathbb{F}), \boldsymbol{c} \leftarrow \mathbb{F}_k^m, \boldsymbol{\Delta_a} \leftarrow \mathbb{F}_k^w) \tag{3}$$

On the other hand, the simulated view is distributed as

$$\left(M \leftarrow \mathcal{M}(1^k, \mathbb{F}), \boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}') + \boldsymbol{e}, \boldsymbol{\Delta_a} \leftarrow \mathbb{F}_k^w\right) \tag{4}$$

where $\boldsymbol{a}' \leftarrow \mathbb{F}_k^w$, and the random variables $\boldsymbol{r}$ and $\boldsymbol{e}$ are distributed as before. By using the pseudorandomness property of M again, we conclude that (4) is computationally indistinguishable from (3), as required. □

This completes the proof of Lemma 6.4. □

6.4 Proof of security against a malicious Bob

We present a simulator that participates in the ideal experiment with the honest Alice. The simulator statistically emulates the view of a malicious Bob $\mathrm{B}^*(\boldsymbol{a}, \boldsymbol{b})$, which we assume to be the output of B^*. The simulator makes a black-box use of B^* in a straight-line way (without rewinding) and uses an oracle to a trusted party, which expects a 2 vectors input $\boldsymbol{v}, \boldsymbol{u} \in \mathbb{F}_k^w$ from it, a field element input $x \in \mathbb{F}_k$ from Alice, and transfers the vector $x\boldsymbol{v} + \boldsymbol{u}$ to the Alice.

Simulator 3 (Bob's Simulator). $\textsc{Sim}_{\mathrm{B}^*}(\boldsymbol{a}, \boldsymbol{b})$

1. *Call Bob's RVOLE Simulator 1 for the adversary* B^**. Recall that the simulator receives from* B^* *a tuple* $(M, \boldsymbol{c}, I)$*, extracts an effective Bob's input* $\boldsymbol{a}'$ *and then queries the ideal* w*-RVOLE functionality with* $\boldsymbol{a}'$*. We respond to this query with a random vector* $\boldsymbol{u} \leftarrow \mathbb{F}_k^w$*. The simulator then generates a vector* $\boldsymbol{d}$ *and a valid CDS vector* $\boldsymbol{z} = (z_0, z_{i,I_i})_{i\in[m]}$ *that uniquely define a secret* Δ_x *that appears explicitly as part of the internal state of the simulator.*
2. *Send the message "commit" to* B^* *and get a field element* $y \in \mathbb{F}_k$ *from him.*
3. *If* $y \neq \Delta_x$ *send an abort message to the trusted party and to* B^* *on behalf of Alice and halt with the output of* B^**.*
   Otherwise, send Δ_x *(as the committed value) to* B^**, get back the vectors* $\boldsymbol{\Delta}_{\boldsymbol{a}}$ *and* $\boldsymbol{v}$*, and send to the trusted party the vectors* $\widetilde{\boldsymbol{a}} = \boldsymbol{\Delta}_{\boldsymbol{a}} + \boldsymbol{a}'$ *and* $\widetilde{\boldsymbol{b}} = \boldsymbol{v} - \boldsymbol{u}$*, terminate with the output of* B^**.*

Lemma 6.7. *For every algorithm* B^* *and ensemble of inputs* $\left\{(\boldsymbol{a}_k, \boldsymbol{b}_k, x_k) \in \mathbb{F}_k^w \times \mathbb{F}_k^w \times \mathbb{F}_k\right\}_{k\in\mathbb{N}}$ *the simulator* $\textsc{Sim}_{\mathrm{B}^*}(\boldsymbol{a}_k, \boldsymbol{b}_k)$ *statistically emulates* B^**.*

Proof. Fix the randomness of B^* and $k \in \mathbb{N}$. Denote by (V_S, w_S) the output of the simulator and Alice in the ideal world and by (V_R, w_R) the output of B^* and Alice in the real world when running the protocol. Denote by $(M, \boldsymbol{c}, I)$ the initial message of B^*. We distinguish between two cases depending on whether $\boldsymbol{c}[I]$ is in $\mathsf{colspan}\left(T[I]\right)$.

Case 1: $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$**.** We prove that the simulation is perfect. First, by the analysis of the RVOLE simulator, the messages of the RVOLE part are distributed identically in both experiments. Moreover, this is true even when we condition on the event that (a) the vector $\boldsymbol{u}$ that is chosen uniformly in the ideal world equals to the vector $x_k\boldsymbol{a}' + \boldsymbol{b}'$ where $\boldsymbol{b}' \leftarrow \mathbb{F}_k^w$ is the vector that is sampled uniformly by Alice in the real world, and (b) that Δ_x as set by the simulator equals to the value $\Delta_x = x_k - x'$ as set in the real experiment for randomly chosen x'. Since the remaining parts of the view can be generated in both experiments by applying the same function to these values it follows that the entire views V_S and V_R are identically distributed (including the values $(y, \boldsymbol{\Delta}_{\boldsymbol{a}}, \boldsymbol{v})$ that are chosen by the adversary). Moving on to analyze the output of Alice, observe that when $y \neq \Delta_x$ in both experiments the output is an "abort" symbol. If $y = \Delta_x$ the output of Alice is

$$w_S = x_k\widetilde{\boldsymbol{a}} + \widetilde{\boldsymbol{b}} = x_k(\boldsymbol{\Delta}_{\boldsymbol{a}} + \boldsymbol{a}') + \boldsymbol{v} - \boldsymbol{u} = x_k(\boldsymbol{\Delta}_{\boldsymbol{a}} + \boldsymbol{a}') + \boldsymbol{v} - x_k\boldsymbol{a}' - \boldsymbol{b}' = x_k\boldsymbol{\Delta}_{\boldsymbol{a}} + \boldsymbol{v} - \boldsymbol{b}' = w_R,$$

where the third equality follows by plugging in $\boldsymbol{u} = x_k\boldsymbol{a}' + \boldsymbol{b}'$.

Case 2: $\boldsymbol{c}[I] \notin \mathsf{colspan}\left(T[I]\right)$**.** Again, since the RVOLE simulator is perfect, the messages of the RVOLE part are distributed identically in both experiments. Moreover, in both experiments, the

CDS secret Δ_x is uniform and is perfectly hidden from the adversary and so, except with probability $1/|\mathbb{F}_k| = \mathsf{negl}(k)$, the adversary fails to guess it and we abort in both experiments.[22] This completes the proof of the lemma. □

7 Actively-Secure VOLE under Fast Pseudorandom Matrix

In this section, we describe a VOLE protocol with full active security based on the modified-RVOLE protocol (Protocol 2). Following the outline in Section 2.3, we begin with some useful observations.

7.1 Useful Observations

More CDS properties. We will make use of the fact that our CDS sends messages only on "zero" inputs. Let $\boldsymbol{z} = (z_0, z_{1,0}, \ldots, z_{m,0})$ be a (possibly malformed) full vector of CDS messages. We say that $\boldsymbol{z}$ is *valid* if for every input I that satisfies the underlying predicate f, the CDS decoder recovers the same secret. We say that $\boldsymbol{z}$ is *honestly generated* if it is generated by invoking the CDS message generator honestly on some random tape and some secret. (By perfect correctness, an honestly generated CDS is always valid.) We assume the existence of an efficient tester $\mathcal{T}$ that rejects every invalid $\boldsymbol{z}$, and accepts every honestly generated $\boldsymbol{z}$. (That is $\mathcal{T}$ is allowed to accept a vector that is not honestly generated as long as it is valid.) Furthermore, if $\mathcal{T}$ accepts then it should be able to recover the secret.[23]

Closer look at cheating Alice. Fix some strategy A^* for a malicious Alice in Protocol 2. Formally, this is a deterministic mapping that takes Alice's inputs $(x, \boldsymbol{b})$, the public matrix M, and Bob's message $\boldsymbol{c}$ and outputs a CDS message vector $\boldsymbol{z} = (z_0, (z_{i,0})_{i\in[m]})$ and a vector $\boldsymbol{d}$ (to be placed on the 1-inputs of the OT). If either $\boldsymbol{z}$ is invalid or $\boldsymbol{d}$ is invalid in the sense that $\boldsymbol{d} \notin \mathsf{colspan}\left((T|\boldsymbol{c})\right)$, we say that A^* *cheats* on $(x, \boldsymbol{b}, M, \boldsymbol{c})$.We show that when Alice does not cheat, her "effective input" can be extracted given her OT messages.

Claim 7.1. *There exists an efficient input-extraction algorithm* R *that takes $M, \boldsymbol{c}, \boldsymbol{z}, \boldsymbol{d}$ and outputs $x', \boldsymbol{b}'$ such that for every (possibly inefficient) A^* for every integer k, every efficient field family $\mathbb{F}$, every Bob's input $\boldsymbol{a} \in \mathbb{F}_k^w$ and every Alice's input $(x, \boldsymbol{b})$, it holds that when Bob's random tape is sampled at random, i.e., $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$, $\boldsymbol{r} \leftarrow \mathbb{F}_k^k$ and $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$, then*

$$\Pr_{M,\boldsymbol{r},\boldsymbol{e}}[(\boldsymbol{z}, \boldsymbol{d}) = A^*(M, (x, \boldsymbol{b}), \boldsymbol{c}) \text{ is valid} \quad \bigwedge \quad x'\boldsymbol{a} + \boldsymbol{b}' \neq \boldsymbol{v}] \leq \mathsf{negl}(k),$$

where $\boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}) + \boldsymbol{e}$, $(x', \boldsymbol{b}') = \mathrm{R}(M, \boldsymbol{c}, \boldsymbol{z}, \boldsymbol{d})$ are the effective inputs of A^* as extracted by R*, and $\boldsymbol{v}$ is the final output of Bob when interacting with A^* as computed in Protocol 2.*

Proof. The algorithm R operates as follows. If $\boldsymbol{z}$ is invalid or $\boldsymbol{d} \notin \mathsf{colspan}\left((T|\boldsymbol{c})\right)$ the algorithm aborts. Otherwise, if the input is valid, R recovers the CDS secret Δ_0 from $\boldsymbol{z}$ by using the input

[22] As mentioned earlier when the field size is small one can always randomly pad Δ_x up length k and derive $\exp(-k)$ deviation error.

[23] The latter property always holds as one can always apply the CDS Reconstruction algorithm with respect to all-zero input. Indeed, since the CDS forms a secret sharing scheme for the predicate $\neg f$ (as discussed in Section 3.4) the all-zero input must satisfy the predicate f (unless f is the trivial 0 predicate).

$I = 0^m$ which satisfies the predicate $f_{T,\boldsymbol{c}}$ for every $\boldsymbol{c}$. Next, R finds a solution $(x_0, \boldsymbol{b}_0, \boldsymbol{r}_0)$ to the system of equations

$$\boldsymbol{d} = T \begin{pmatrix} \boldsymbol{r}_0 \\ \boldsymbol{b}_0 \end{pmatrix} + x_0 \boldsymbol{c},$$

sets $x'_0 = x_0 + \Delta_0$, and outputs $(x'_0, \boldsymbol{b}_0)$ as Alice's effective inputs.

Analysis: Parse $\boldsymbol{z} = (z_0, z_{1,0}, \ldots, z_{m,0})$ and let $I = I(\boldsymbol{e})$ and let us assume that A^* outputs valid vectors. Recall that Bob's output is computed as follows. First, Bob reconstructs the CDS secret Δ_1 by decoding $\mathsf{Dec}(I, (z_0, (z_{i,I_i})_{i\in[m]}))$. Since $\boldsymbol{z}$ is valid, decoding succeeds, and Δ_1 equals to the value Δ_0 that is recovered by R. Next, Bob computes a solution $(\boldsymbol{r}_1, \boldsymbol{v}_1)$ to the system

$$\boldsymbol{d}[I] = T[I] \cdot \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{v}_1 \end{pmatrix}, \tag{5}$$

and aborts if no such solution is found. Let us condition on the event that (5) has at most one solution. By Assumption 4.1 (property 3), this event holds except with negligible probability (over the choice of M and I). Now since $\boldsymbol{c}[I] = T \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{a} \end{pmatrix}$, the vector $\begin{pmatrix} \boldsymbol{r}_0 \\ \boldsymbol{b}_0 \end{pmatrix} + x_0 \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{a} \end{pmatrix}$ also solves (5). By uniqueness, we conclude that $\boldsymbol{r}_1 = \boldsymbol{r}_0 + x_0 \boldsymbol{r}$ and $\boldsymbol{v}_1 = \boldsymbol{b}_0 + x_0 \boldsymbol{a}$. Finally, recall that Bob's final output $\boldsymbol{v}$ is taken to be $\boldsymbol{v_1} + \Delta_1 \boldsymbol{a}$, and is therefore equal to

$$\boldsymbol{b}_0 + x_0 \boldsymbol{a} + \Delta_1 \boldsymbol{a} = \boldsymbol{b}_0 + x'_0 \boldsymbol{a},$$

as required. □

We can now prove the following lemma. Let us denote Protocol 2 by Π.

Lemma 7.2. *There exists a simulator* $\textsc{Sim}'(x, \boldsymbol{b})$ *that makes a black-box use of* A^* *and simulates* Π *whenever* A^* *does not cheat. Formally, for every sequence of inputs* $((x_k, \boldsymbol{b}_k), \boldsymbol{a}_k)$ *it holds that the ensemble*

$$\left[\textsc{Real}_{A^*,\Pi}((x_k, \boldsymbol{b}_k), \boldsymbol{a}_k) \quad | \quad A^* \textit{ doesn't cheat} \right]$$

is computationally indistinguishable from the ensemble

$$\left[\textsc{Ideal}_{\textsc{Sim},f_k}(\boldsymbol{a}_k, \boldsymbol{b}_k) \quad | \quad \textsc{Sim} \textit{ doesn't fail} \right].$$

Proof. Given $(x_k, \boldsymbol{b}_k)$ and a matrix M that was sampled from $\mathcal{M}(1^k, \mathbb{F})$ (see Remark 5), the simulator samples a random vector $\boldsymbol{c} \in \mathbb{F}_k^m$, feeds A^* with $(x, \boldsymbol{b}, M, \boldsymbol{c})$ and receives an output $(\boldsymbol{d}, \boldsymbol{z})$ (to be sent over the OT channels). The simulator checks the validity of these vectors, and if A^* cheats the simulator aborts with a special failure symbol. Otherwise, A^* recovers the effective inputs $(x', \boldsymbol{b}')$ by using the input extraction algorithm R, and passes $(x', \boldsymbol{b}')$ to the ideal functionality who passes the output to Bob.

To analyze the simulator, fix an input $\boldsymbol{a}$ for Bob, and consider the hybrid experiment $\mathcal{H}$ where $\boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}) + \boldsymbol{e}$ for some properly sampled $\boldsymbol{r}$ and $\boldsymbol{e}$. The rest of the simulation is performed as above. By Claim 7.1, conditioned on not aborting the output of the hybrid experiment, $(\boldsymbol{c}, \boldsymbol{a}x + \boldsymbol{b}')$ is statistically close to the output of the real experiment.

On the other hand, by Assumption 4.1, the output of $\mathcal{H}$ is computationally indistinguishable from the output of the simulated experiment. Indeed, if a distinguisher D can distinguish between

the two distributions, we can break Assumption 4.1 as follows. Given a matrix M and a challenge $\boldsymbol{u}$ which is either random or pseudorandom, set $\boldsymbol{c} = \boldsymbol{u} + 0^u \circ \mathsf{Ecc}(\boldsymbol{a})$, and continue exactly as in the standard simulation. That is, compute $(\boldsymbol{d}, \boldsymbol{z}) := A^*(x, \boldsymbol{b}, M, \boldsymbol{c})$, output "abort" if the outcome is invalid, and otherwise set $(x', \boldsymbol{b}') := \mathrm{R}(M, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{z})$, and output the pair $(\boldsymbol{c}, x'\boldsymbol{a} + \boldsymbol{b}')$. The output of this procedure is distributed identically to the simulator when $\boldsymbol{u}$ is uniform and identically to $\mathcal{H}$ when $\boldsymbol{u}$ is pseudorandom. Thus, we can use D to distinguish between the two cases, in contradiction to Assumption 4.1. □

The next crucial observation is that there exists a strategy for Bob that detects (with probability 0.5) whether Alice cheats. Indeed, as explained in the introduction, in the OT phase Bob can toss a coin and ask with probability 1/2 to receive the vector $\boldsymbol{d}$ (by using $I = 1^m$) and with probability 1/2 the vector $\boldsymbol{z}$ (by using $I = 0^m$) and check validity. When running in this "detection mode" Bob effectively gives up on the computation and just verifies whether Alice misbehaves or not. Note that Bob's decision is taken only in the OT phase and is hidden from Alice, and so effectively Alice first "commits" to strategy (cheat or not), and only then Bob decides whether to "call her bluff". Furthermore, even when Bob acts as a detector, we can fully simulate his view (since the protocol is actively-secure against any deviation of Bob). We will exploit this property to obtain a "silent" cut-and-choose version of the protocol as follows.

7.2 The VOLE3 protocol

Let Π denote Protocol 2 instantiated with width $w(k) = O(k^3)$ and recall that Π makes a call to an $m = m(k)$-batch OT channel. The new protocol (hereafter denoted as VOLE3) realizes VOLE with width $W = W(k)$ (for some value that will be determined later) by making $t = t(k)$ calls to Π, and by "opening" $p = p(k)$ sessions for detecting a potential cheating by Alice. In addition to these parameters, we let $s = s(k) = t(k) - p(k)$ denote the number of remaining "un-opened" sessions, and let $\ell = \ell(k)$ be a leakage parameter. The product $\ell p/t$ should be polynomial in the security parameter k and, for efficiency purposes, s should be $\Omega(t)$. For example, set $t = k$, $p = \ell = k^{0.9}$ and $s = k - k^{0.9}$. Again, we make use of ideal commitments which can be realized based on OT channels(See Remark 6).

Linear-time resilient functions. We will need a linear mapping $\mathsf{Ext} : \mathbb{F}^{sw} \to \mathbb{F}^{(1-\beta)sw}$ where $\beta = \beta(k) < 1$ is bounded away from 1, with the following properties: (1) Ext should be computable in linear arithmetic time (i.e., by making $O(sw)$ operations); and (2) The distribution $\mathsf{Ext}(X)$ should be uniform whenever the input X is uniform except for at most $\ell' = \ell(k)w + 1 = o(sw)$ entries that may be arbitrarily fixed. Formally, for every ℓ'-subset $L \subset [sw]$ and fixing $(X_i)_{i \in L} \in \mathbb{F}^{\ell'}$ if $(X_i)_{i \notin L}$ is uniform over $\mathbb{F}^{sw-\ell'}$, the output $\mathsf{Ext}(X_1, \ldots, X_{sw})$ is uniform over $\mathbb{F}^{(1-\beta)sw}$. Such functions are known as *ℓ'-resilient functions* [CGH$^+$85] and can also be viewed as perfect deterministic extractors for bit-fixing sources. One can realize such functions, with the desired parameters, based on linear-time encodable error-correcting codes with rate $1-\beta$ and distance $\ell'+1$ (see [CGH$^+$85] and [Dru13, Theorem 3.1.7]). The width of VOLE3 is taken to be the output length of Ext, i.e., $W(k) = (1-\beta)sw$.

Protocol 4 (VOLE3 protocol). *Upon initialization, bob samples $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$ and sends it to Alice. The input of Bob is a pair of vectors $\boldsymbol{g}, \boldsymbol{f} \in \mathbb{F}_k^W$ and the input of Alice is $x \in \mathbb{F}_k$.*

1. ***Bob:*** *Invokes t independent parallel sessions of Π with the matrix M as a public parameter, where the jth private input is a random vector $\boldsymbol{a}_j \leftarrow \mathbb{F}_k^w$. For each such session $j \in [t]$, Bob computes the first-round message $\boldsymbol{c}_j$ as in Step 1 of Π, and sends $\boldsymbol{c}_j$. Let I_j denote the (vector representation of the) set of clean coordinates in $\boldsymbol{c}_j$.*
2. ***Alice:*** *Samples $x^* \leftarrow \mathbb{F}_k$. For every $j \in [t]$, Alice sets her inputs for the jth session to be $(x_j, \boldsymbol{b}_j)$ where $x_j = x^*$ and $\boldsymbol{b}_j \leftarrow \mathbb{F}_k^w$, she samples a random tape for the jth session, sends the corresponding offline CDS message $z_{i,0}$ to Bob, and computes the vectors $(\boldsymbol{z}_j, \boldsymbol{d}_j)$ that will be sent in the OT phase of the jth session (by following Steps 2 and 3 of Π). Here we view Alice's input to the m-batch-OT as a pair of m-long vectors.*
3. ***OT-phase:*** *Bob samples a random p-subset $P \subset [t]$ of sessions that will be "opened". For each $j \in [t]$ in parallel, the parties invoke the m-batch OT channel of the jth session. Alice's input is $(\boldsymbol{z}_j, \boldsymbol{d}_j)$. If $j \notin P$ Bob's input is I_j; Otherwise, Bob samples a random bit $\sigma_j \leftarrow \{0,1\}$ and uses a trivial selection vector $I'_j := \sigma_j^m \in \{0^m, 1^m\}$.*
4. ***Bob:*** *If cheating is detected in one of the "opened copies" $j \in P$ (i.e., if the received vector is invalid), Bob aborts. Otherwise, let $S = [t] \setminus P$ denote the set of unopened copies. For every $j \in S$, Bob recovers the output $\boldsymbol{v}_j \in \mathbb{F}_k^w$ of the jth session just like in Step 5 in Π (hereafter referred to as Π_5). If the output is "abort" Bob sets $\boldsymbol{v}_j = 0^w$.*
5. ***Sub-protocol:*** *To verify that Alice's inputs $(x_j)_{j \in S}$ are all equal, the parties do:*
   - ***Bob:*** *Computes the sum δ of all the last elements of the vectors $(\boldsymbol{a}_j \in \mathbb{F}_k^w)_{j \in S}$, i.e., $\delta := \sum_{j \in S} \boldsymbol{a}_j[w]$ and sends to Alice the pair (S, δ). (Bob challenges Alice to compute the sum $\sum_{j \in S} \boldsymbol{v}_j[w]$.)*
   - ***Alice:*** *Sends $\lambda = x^*\delta + \sum_{j \in S} \boldsymbol{b}_j[w]$ over the ideal commitment channel which delivers a "commit" message to Bob.*
   - ***Bob:*** *Given a "commit" message, sends the value $\lambda' := \sum_{j \in S} \boldsymbol{v}_j[w]$.*
   - ***Alice:*** *decommits by sending an "open" message to the commitment channel if $\lambda' = \lambda$, else Alice aborts.*
   - ***Bob:*** *Halts with an abort symbol if Alice does not open the commitment or if the decommitment $\lambda \neq \lambda'$, and continues otherwise.*
6. ***Alice:*** *Sends $\Delta = x - x^*$.*
7. ***Bob:*** *Aligns the results of the unopened sessions vectors by setting*

$$\boldsymbol{u}_j := \boldsymbol{v}_j + \Delta \boldsymbol{a}_j, \quad \forall j \in S,$$

   concatenates the vectors $(\boldsymbol{a}_j)_{j \in S}$ to a single vector $\boldsymbol{a}_S \in \mathbb{F}_k^{sw}$ and the vectors $(\boldsymbol{u}_j)_{j \in S}$ to a single vector $\boldsymbol{u}_S \in \mathbb{F}_k^{sw}$, and extracts the vectors

$$\boldsymbol{a}' := \mathsf{Ext}(\boldsymbol{a}_S), \quad \boldsymbol{u}' := \mathsf{Ext}(\boldsymbol{u}_S).$$

   Bob sends to Alice the vectors $\boldsymbol{\alpha} := \boldsymbol{f} - \boldsymbol{a}'$ and $\boldsymbol{h} := \boldsymbol{u}' + \boldsymbol{g}$.
8. ***Alice:*** *Concatenates $(\boldsymbol{b}_j)_{j \in S}$ to a vector $\boldsymbol{b}_S \in \mathbb{F}_k^{sw}$, extracts $\boldsymbol{b}' := \mathsf{Ext}(\boldsymbol{b}_S)$, and outputs $\boldsymbol{h} + x\boldsymbol{\alpha} - \boldsymbol{b}'$.*

Analysis. The protocol has an arithmetic complexity of $O(tw) = O(sw) = O(W)$, as required. The communication complexity is dominated by the complexity of Steps 1–3 and Step 7 which is $t \cdot C_\Pi(w) + 2W = tO(w) + O(W) = O(W)$ where $C_\Pi(w)$ is the communication complexity of Π over width w. (The communication in Steps 4-6 consists of $O(s)$ bits and $O(k)$ field elements for realizing the commitment channel via OT). By employing extractors that shrink their input by a factor of $1-\beta$ for arbitrarily small constant β (e.g., based on the codes of [GI05]), we can take $W = (1-\beta)(1-o(1))tw$. Recalling that $C_\Pi(w)$ approaches to $3w$, the total communication of VOLE3 approaches to $t3w + 2W = 5W/(1-\beta-o(1))$, i.e., the asymptotic rate approaches to $1/5$. [24]The simulators for Alice and Bob appear in Sections 7.3 and 7.4, leading to Theorem 2.1.

Theorem 7.3 (Theorem 2.1 restated)**.** *Under Assumption 4.1, Protocol 4 (VOLE3) realizes the VOLE functionality of width W over $\mathbb{F}$ with arithmetic complexity of $O(W)$ in the OT-hybrid model. The protocol is statistically-secure against an active sender Bob with negligible deviation error and computationally secure against an active receiver Alice.*

7.3 Simulating Alice

We simulate a malicious Alice $A^*(x)$ as follows.

Simulator 4 (**Alice's Simulator for VOLE3**)**.** $\text{SIM}_{A^*}(x)$*: Given an input x and a matrix M that was sampled from $\mathcal{M}(1^k, \mathbb{F})$ (see Remark 5), the simulator invokes the protocol with Alice $A^*(x)$ by playing Bob honestly with the following modifications.*

- ***At the end of Step 4 do:***
  Partition the set S of unopened *sessions into the set $L \subset S$ of unopened sessions in which Alice* cheats *and the set $H = S \setminus L$ of unopened sessions in which Alice does not cheat. If L is larger than ℓ, we abort. Else, extract Alice's "effective input" $x_j, \boldsymbol{b}_j$ for every $j \in H$ by using the input-extraction algorithm* R *from Claim 7.1. Set*

  $$v_j := x_j \hat{\boldsymbol{a}}_j + \boldsymbol{b}_j$$

  where $\hat{\boldsymbol{a}}_j \leftarrow \mathbb{F}_k^w$ is sampled independently at random. Also, from this point all occurrences of $\boldsymbol{a}_j, j \in H$ will be replaced with $\hat{\boldsymbol{a}}_j$. (So $\boldsymbol{a}_j, j \in H$ are being used only to generate $\boldsymbol{c}_j$ in Step 1.)
- ***Instead of Step 7 do:***
  If $x_j \neq x_{j'}$ for some $j, j' \in H$, abort.
  Set $x' = x_j + \Delta$ for some $j \in H$, call the ideal W-width VOLE functionality where Alice's input is x', and let $\boldsymbol{y}$ denote the output. Send to Alice the vectors

  $$\boldsymbol{\alpha} = -\boldsymbol{a}', \quad \boldsymbol{h} = \boldsymbol{u}' + \boldsymbol{y},$$

  where $\boldsymbol{a}'$ and $\boldsymbol{u}'$ are computed as in the protocol (except that, for $j \in H$, the vector $\boldsymbol{a}_j$ is replaced with $\hat{\boldsymbol{a}}_j$). That is, we play like Bob with $\boldsymbol{f} = \boldsymbol{0}$ and $\boldsymbol{g} = \boldsymbol{y}$.

[24]A useful relaxation of VOLE (see [BCG+19b]) assumes that $\boldsymbol{f}$ is chosen at random for an honest Bob and arbitrarily by a malicious Bob. To implement this variant, we can let $\boldsymbol{f} = \boldsymbol{a}'$ set $\boldsymbol{\alpha} = 0^W$ and omit it from the communication, thus improving the rate to $1/4$.

We note that when the protocol makes a call to batch-OT we simply receive from malicious Alice the inputs for the batch-OT. (Since Alice is the sender, we do not need to provide any outcome for these calls.)

To analyze the simulator, fix a sequence of inputs $(x_k)_{k\in\mathbb{N}}$ and $(\boldsymbol{f}_k, \boldsymbol{g}_k)_{k\in\mathbb{N}}$ for Alice and Bob respectively. We will show that Alice's simulated view in the ideal execution is indistinguishable from its view in the real execution. From now on, we omit the subscripts from the inputs and take $x = x_k, \boldsymbol{f} = \boldsymbol{f}_k, \boldsymbol{g} = \boldsymbol{g}_k$) where k is the security parameter. We proceed with a sequence of hybrid arguments. (For the convenience of the reader, the hybrids also appear in Tables 1 and 2.)

The hybrid $\mathcal{H}_0$. This hybrid is identical to the real execution except that in Step 4, we partition the set S of unopened sessions into the set $L \subset S$ of unopened sessions in which Alice *cheats* and the set $H = S \setminus L$ of unopened sessions in which Alice does not cheat, and abort if L is larger than ℓ. (Note that the values $\boldsymbol{v}_j$ are defined as in the real execution for both $j \in H$ and $j \in L$.)

Claim 7.4. *The output of the real execution and the output of $\mathcal{H}_0$ are statistically indistinguishable.*

Proof. A deviation occurs only if A^* cheats on at least ℓ instances without being detected at all by the set P. We show that this event happens with negligible probability. Fix all the randomness in the first 2 steps and condition on the event that A^* cheats in at least ℓ instance . Let χ_i denote an indicator random variable that takes the value 1 if the ith cheating is being detected where the randomness is induced by the choise of P and $\sigma = (\sigma_j)_{j\in P}$. Since P and σ are statistically independent of the view of A^*, it holds that, for every i, χ_i takes the value one with probability $p/(2t)$. Furthermore, for every i,

$$\Pr[\chi_i = 1 | \chi_1 = \cdots = \chi_{i-1} = 0] \geq \Pr[\chi_i = 1]$$

since we sample P without replacement. Hence,

$$\Pr[\chi_1 = \cdots = \chi_\ell = 0] \leq (1 - p/(2t))^\ell \leq \exp(-\Omega(\ell p/t)),$$

which is negligible in k since $\ell p/t$ is polynomial in k. □

The hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$, except that for every $j \in H$, we extract Alice's "effective input" $x_j, \boldsymbol{b}_j$ by using the input-extraction algorithm R from Claim 7.1 and set $v_j := x_j \boldsymbol{a}_j + \boldsymbol{b}_j$. (For $j \in S \setminus H$, we set $\boldsymbol{v}_j$ exactly as in the real execution.) By Claim 7.1, it follows that:

Claim 7.5. *The Hybrid $\mathcal{H}_0$ is statistically indistinguishable from the Hybrid $\mathcal{H}_1$.*

The hybrid $\mathcal{H}_2$. This hybrid is identical to $\mathcal{H}_1$ except that in Step 4, for every $j \in H$, we resample the vector $\hat{\boldsymbol{a}}_j \leftarrow \mathbb{F}_k^w$. These vectors are being used from now on instead of the original ones, including in the definition of $\boldsymbol{v}_j, j \in H$.

Claim 7.6. *The Hybrid $\mathcal{H}_1$ is computationally indistinguishable from $\mathcal{H}_2$.*

Proof. We use sub-hybrids. For $h \in [t+1]$, let $\mathcal{H}_{1,h}$ denote the hybrid which is identical to $\mathcal{H}_2$ except that $\boldsymbol{a}_j$ is resampled only if $j < h$ and $j \in H$. Clearly, $\mathcal{H}_{1,1}$ is identical to $\mathcal{H}_1$ and $\mathcal{H}_{t+1}$ is identical to $\mathcal{H}_2$. If the claim does not hold then there exists some index $h \in [t]$, and an efficient distinguisher D and a non-negligible function ϵ such that

$$\Pr[D(\mathcal{H}_{1,h+1}) = 1] - \Pr[D(\mathcal{H}_{1,h}) = 1] > \epsilon.$$

In this case, we can break Assumption 4.1 as follows.

Given M and a challenge $\boldsymbol{u}$ which is either random or pseudorandom, sample two random vectors $\boldsymbol{a}^0$ and $\boldsymbol{a}^1$ and set $\boldsymbol{c} = \boldsymbol{u} + 0^u \circ \mathsf{Ecc}(\boldsymbol{a}_h^0)$. Then invoke $\mathcal{H}_{1,h}$ with the matrix M and with $\boldsymbol{c}_h$ set to $\boldsymbol{c}$. If $j \notin H$ output 0 and terminate. Otherwise, in the resampling step, toss a coin $\sigma \leftarrow \{0,1\}$ and set $\boldsymbol{a}_h$ to $\boldsymbol{a}^\sigma$. At the end, feed the output of the procedure to the distinguisher D and output "pseudorandom" if D's output equals to σ.

The analysis is standard. Assume that $\boldsymbol{u}$ is pseudorandom. Observe that when $\sigma = 0$ (resp., $\sigma = 1$) we perfectly emulate the distribution of $\mathcal{H}_h$ (resp., $\mathcal{H}_{h+1}$) conditioned on the event that $h \notin H$. Next, note that when $h \notin H$, the original hybrids $\mathcal{H}_{1,h}$ and $\mathcal{H}_{1,h+1}$ are identical and so the distinguisher has zero-advantage in this case. It follows that when $\boldsymbol{u}$ is pseudorandom, the adversary guesses σ with probability $0.5+\epsilon/2$. On the other hand, when $\boldsymbol{u}$ is random, the distribution handed to D is independent of σ and so the guessing probability is 0.5. Thus, we distinguish random from pseudorandom with non-negligible advantage of $\epsilon/2$. The claim follows. □

The hybrid $\mathcal{H}_3$. This hybrid is identical to $\mathcal{H}_2$ except that at Step 7, we abort if $x_j \neq x_{j'}$ for some $j, j' \in H$.

Claim 7.7. *The Hybrid $\mathcal{H}_2$ is statistically indistinguishable from the Hybrid $\mathcal{H}_3$.*

Proof. We show that conditioned on the event that $x_j \neq x_{j'}$ for some $j, j' \in H$, the probability that $\lambda = \sum_{j \in S} \boldsymbol{v}_j[w]$ is at most $1/|\mathbb{F}_k| = \mathsf{negl}(k)$. Let us arbitrarily fix all the randomness in the experiment except for the resampled vectors $\boldsymbol{a}_j, \boldsymbol{a}_{j'}$. First observe that, at the beginning of Step 6, $\boldsymbol{a}_j, \boldsymbol{a}_{j'}$, are uniform even conditioned on the view of A^* and on the values of $(x_j, \boldsymbol{b}_j)_{j \in S}$. Hence, the scalars $\boldsymbol{a}_j[w], \boldsymbol{a}_{j'}[w]$ are also uniform conditioned on all these values. Let us further condition on

$$\delta = \boldsymbol{a}_j[w] + \boldsymbol{a}_{j'}[w] + C,$$

where $C = \sum_{i \neq j,j'} \boldsymbol{a}_i[w]$ is a fixed value. Let $z_j = x_j + \Delta$ and $z_{j'} = x_{j'} + \Delta$, and consider the following linear combination of $\boldsymbol{a}_j[w], \boldsymbol{a}_{j'}[w]$:

$$z_j \cdot \boldsymbol{a}_j[w] + z_{j'} \cdot \boldsymbol{a}_{j'}[w]. \tag{6}$$

Since $z_j \neq z_{j'}$, this linear combination is linearly independent from the one induced by δ, and so (6) is uniformly distributed over $\mathbb{F}_k$. Recall that for $i \in H$, we can write $\boldsymbol{v}_i[w]$ as $x_i \boldsymbol{a}_i[w] + \boldsymbol{b}_i[w]$, and therefore,

$$\Pr_{\boldsymbol{a}_j, \boldsymbol{a}_{j'}} \left[\lambda = \sum_{i \in S} \boldsymbol{v}_i[w] \right] = \Pr_{\boldsymbol{a}_j, \boldsymbol{a}_{j'}} \left[\lambda = z_j \boldsymbol{a}_j[w] + z_{j'} \boldsymbol{a}_{j'}[w] + C' \right] \leq 1/|\mathbb{F}_k|,$$

where $C' = \boldsymbol{b}_j[w] + \boldsymbol{b}_{j'}[w] + \sum_{i \neq j,j'} \boldsymbol{v}_i[w]$. The inequality follows by noting that, due to our conditioning, C' and λ are fixed. (Indeed, λ depends only on the view of A^*.) The claim follows. □

Claim 7.8. *The output of $\mathcal{H}_3$ and the output of the simulator are identically distributed.*

Proof. Fix all the randomness in both $\mathcal{H}_3$ and in the simulation except for the vectors $\boldsymbol{a}_j, j \in H$. Let us further condition on some fixed value of δ, and note that this means that $(\boldsymbol{a}_j, j \in H)$ is uniform except for the last entry of the last vector $\boldsymbol{a}_{j_0}$ (which is taken to be $\delta - \sum_{j_0 \neq j \in H} \boldsymbol{a}_j[w]$). We show that the pair $(\boldsymbol{\alpha}, \boldsymbol{h})$ as distributed in $\mathcal{H}_3$ is identically distributed to the pair $(\boldsymbol{\alpha}, \boldsymbol{h})$ as

#	$\mathcal{H}_0$	$\mathcal{H}_1$	$\mathcal{H}_2$
1	$\forall j \in [t], \boldsymbol{c}_j := E_{\boldsymbol{r}_j}(\boldsymbol{a}_j) + \boldsymbol{e}_j, I_j := I(\boldsymbol{a}_j)$ where $\boldsymbol{r}_j, \boldsymbol{a}_j, \boldsymbol{e}_j$ are random		
2	$(\boldsymbol{z}_j, \boldsymbol{d}_j)_{j\in[t]} := A^*(x, M, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_t)$		
3	Sample p-subset $P \subset [t]$, $\sigma_j \leftarrow \{0,1\}, \forall j \in P$,		
4	**Abort** if $\exists j \in P$: ($\sigma_j = 0$ AND $\boldsymbol{z}_j$ is invalid) OR ($\sigma_j = 1$ AND $\boldsymbol{d}_j$ is invalid)		
	Let $S := [t] \setminus P$		
	Let $L := \{j \in S : (\boldsymbol{z}_j, \boldsymbol{d}_j) \text{ is invalid}\}$		
	Abort if $\|L\| > \ell$		
	Let $H := S \setminus L$		Resample $(\boldsymbol{a}_j)_{j\in H}$
	$\forall j \in L, \boldsymbol{v}_j := \Pi_5(I_j, \boldsymbol{z}_j[\overline{I}_j], \boldsymbol{d}_j[I_j])$		
	$\forall j \in H, \boldsymbol{v}_j := \Pi_5(I_j, \boldsymbol{z}_j[\overline{I}_j], \boldsymbol{d}_j[I_j])$	$\forall j \in H$, $(x_j, \boldsymbol{b}_j) := \mathrm{R}(M, \boldsymbol{c}_j, \boldsymbol{z}_j, \boldsymbol{d}_j)$ $v_j := x_j \boldsymbol{a}_j + \boldsymbol{b}_j$	
5	$\delta := \sum_{j\in S} \boldsymbol{a}_j[w]$		
	Get input to commitment $\lambda := A^*(\delta, S)$		
	Set $\lambda' := \sum_{j\in S} \boldsymbol{v}_j[w]$		
	Abort if $A^*(\lambda') \neq \mathsf{open}$ or $\lambda \neq \lambda'$		
6	Get $\Delta \leftarrow A^*$		
7	$\forall j \in S, \boldsymbol{u}_j := \boldsymbol{v}_j + \Delta \boldsymbol{a}_j$		
	$\boldsymbol{a}_S := (\boldsymbol{a}_j)_{j\in S}, \boldsymbol{u}_S := (\boldsymbol{u}_j)_{j\in S}$		
	$\boldsymbol{a}' := \mathsf{Ext}(\boldsymbol{a}_S)$, $\boldsymbol{u}' := \mathsf{Ext}(\boldsymbol{u}_S)$		
	Set $\boldsymbol{\alpha} := \boldsymbol{f} - \boldsymbol{a}'$ and $\boldsymbol{h} := \boldsymbol{u}' + \boldsymbol{g}$		
	Output $A^*(\boldsymbol{\alpha}, \boldsymbol{h})$		

Table 1: The hybrids $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2$. Random elements are chosen according to the distributions defined in the protocol (e.g., $\boldsymbol{r}_j$ is uniform, $\boldsymbol{e}_j$ is sampled according to the noise distribution and P is a random p-subset of $[t]$). We write $\overline{I}$ to denote the complement of a set I. Each column is devoted to a different hybrids. In each hybrid, we color the lines that deviate from the original protocol and, to avoid clutter, omit lines that are identical to the previous hybrid.

sampled by the simulator. By the properties of Ext, it follows that $\boldsymbol{a}'$ is uniform, and so $\boldsymbol{\alpha}$ is also uniform in both experiments. Let us condition on some fixing of $\boldsymbol{\alpha}$ in both experiments, and analyze the conditional distribution of $\boldsymbol{h}$.

In both experiments it holds that

$$\begin{aligned}\boldsymbol{u}' &= \mathsf{Ext}((\boldsymbol{u}_j)_{j\in S}) \\ &= \mathsf{Ext}(((x_j + \Delta)\boldsymbol{a}_j + \boldsymbol{b}_j)_{j\in S}) \\ &= x'\mathsf{Ext}((\boldsymbol{a}_j)_{j\in S}) + \mathsf{Ext}((\boldsymbol{b}_j)_{j\in S}) \\ &= x'\boldsymbol{a}' + \mathsf{Ext}((\boldsymbol{b}_j)_{j\in S}),\end{aligned}$$

where x' is defined as in the simulation, and for $j \in L$, we let $\boldsymbol{b}_j = \boldsymbol{v}_j - x'\boldsymbol{a}_j$. Let $\boldsymbol{b}' := \mathsf{Ext}((\boldsymbol{b}_j)_{j\in S})$. In $\mathcal{H}_3$, it holds that

$$\boldsymbol{h} = \boldsymbol{u}' + \boldsymbol{g} = x'\boldsymbol{a}' + \boldsymbol{b}' + \boldsymbol{g} = x'(\boldsymbol{f} - \boldsymbol{\alpha}) + \boldsymbol{b}' + \boldsymbol{g} = (x'\boldsymbol{f} + \boldsymbol{g}) - x'\boldsymbol{\alpha} + \boldsymbol{b}',$$

similarly in the simulation we have

$$\boldsymbol{h} = \boldsymbol{u}' + \boldsymbol{y} = x'\boldsymbol{a}' + \boldsymbol{b}' + x'\boldsymbol{f} + \boldsymbol{g} = x'\boldsymbol{f} + \boldsymbol{g} - x'\boldsymbol{\alpha} + \boldsymbol{b}',$$

as required. □

#	$\mathcal{H}_2$	$\mathcal{H}_3$	$\text{SIM}(M, x)$
1	$\forall j \in [t], \boldsymbol{c}_j := E_{\boldsymbol{r}_j}(\boldsymbol{a}_j) + \boldsymbol{e}_j, I_j := I(\boldsymbol{a}_j)$ where $\boldsymbol{r}_j, \boldsymbol{a}_j, \boldsymbol{e}_j$ are random		
2	$(\boldsymbol{z}_j, \boldsymbol{d}_j)_{j\in[t]} := A^*(x, M, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_t)$		
3	Sample p-subset $P \subset [t]$, $\sigma_j \leftarrow \{0,1\}, \forall j \in P$,		
4	**Abort** if $\exists j \in P$: ($\sigma_j = 0$ AND $\boldsymbol{z}_j$ is invalid) OR ($\sigma_j = 1$ AND $\boldsymbol{d}_j$ is invalid) Let $S := [t] \setminus P$ Let $L := \{j \in S : (\boldsymbol{z}_j, \boldsymbol{d}_j)$ is invalid$\}$ **Abort** if $\lvert L\rvert > \ell$ Let $H := S \setminus L$, resample $(\boldsymbol{a}_j)_{j\in H}$ $\forall j \in L, \boldsymbol{v}_j := \Pi_5(I_j, \boldsymbol{z}_j[\overline{I}_j], \boldsymbol{d}_j[I_j])$ $\forall j \in H$, $(x_j, \boldsymbol{b}_j) := \mathrm{R}(M, \boldsymbol{c}_j, \boldsymbol{z}_j, \boldsymbol{d}_j)$ $v_j := x_j \boldsymbol{a}_j + \boldsymbol{b}_j$		
5	$\delta := \sum_{j\in S} \boldsymbol{a}_j[w]$ Get input to commitment $\lambda := A^*(\delta, S)$ Set $\lambda' := \sum_{j\in S} \boldsymbol{v}_j[w]$ **Abort** if $A^*(\lambda') \neq \mathsf{open}$ or $\lambda \neq \lambda'$		
6	Get $\Delta \leftarrow A^*$		
7	$\forall j \in S, \boldsymbol{u}_j := \boldsymbol{v}_j + \Delta \boldsymbol{a}_j$	**Abort** if $\exists j, j' \in H$: $x_j \neq x_{j'}$	$x' := x_j + \Delta$ for some $j \in H$ $\boldsymbol{y} := \mathsf{VOLE}_{\boldsymbol{f},\boldsymbol{g}}(x')$ $= x'\boldsymbol{f} + \boldsymbol{g}$
	$\boldsymbol{a}_S := (\boldsymbol{a}_j)_{j\in S}, \boldsymbol{u}_S := (\boldsymbol{u}_j)_{j\in S}$ $\boldsymbol{a}' := \mathsf{Ext}(\boldsymbol{a}_S)$, $\boldsymbol{u}' := \mathsf{Ext}(\boldsymbol{u}_S)$ Set $\boldsymbol{\alpha} := \boldsymbol{f} - \boldsymbol{a}'$ and $\boldsymbol{h} := \boldsymbol{u}' + \boldsymbol{g}$ Output $A^*(\boldsymbol{\alpha}, \boldsymbol{h})$		$\boldsymbol{\alpha} := -\boldsymbol{a}'$, $\boldsymbol{h} = \boldsymbol{u}' + \boldsymbol{y}$

Table 2: The simulator and some hybrids. We follow the conventions of Table 1. The hybrid $\mathcal{H}_2$ is repeated for clarity.

7.4 Simulating Bob

Given an input $(\boldsymbol{f}, \boldsymbol{g})$ and access to the W-VOLE ideal functionality, we simulate a malicious Bob $B^*(\boldsymbol{f}, \boldsymbol{g})$ via a straight-line black-box simulator as follows. (If B^* aborts at some point we end the simulation and send an abort symbol to the VOLE-oracle.)

Simulator 5 (Bob's Simulator for VOLE3). $\text{SIM}_{B^*}(\boldsymbol{f}, \boldsymbol{g})$*:*

1. *Call* $B^*(\boldsymbol{f}, \boldsymbol{g})$ *and get a matrix* M*, and, for every session* $j \in [t]$ *of* Π*, a first round message* $\boldsymbol{c}_j$ *and an OT selection vector* I_j*.*
2. *For* $j \in [t]$ *sample a random output* $\boldsymbol{v}_j \leftarrow \mathbb{F}_k^w$ *for the* j*th session.*
3. *For every* $j \in [t]$*, invoke Bob's simulator for* Π *where Bob's first message is taken to be* $(M, \boldsymbol{c}_j, I_j)$*. Recall that the simulator first extracts an effective input* $\boldsymbol{a}_j$ *and then queries the ideal* w*-RVOLE functionality with* $\boldsymbol{a}_j$*. Respond to this query with* $\boldsymbol{v}_j$*, and pass the simulator's final output* $(\boldsymbol{z}_j[i])_{i \notin I_j}, (\boldsymbol{d}_j[i])_{i \in I_j}$ *to* B^* *as the output of the OT for the* j*th session.*
4. *If* B^* *outputs an abort symbol, send an abort symbol to the ideal functionality and terminate the simulation with Bob's output.*
5. *Emulate the "verification" sub-protocol as follows:*
   - B^* *sends* (δ, S)*.*
   - *Send to* B^* *a* commit *message on behalf of the commitment channel.*
   - B^* *sends* λ'*.*
   - *If* $\lambda' \neq \sum_{j \in S} \boldsymbol{v}_j[w]$ *or* $\delta \neq \sum_{j \in S} \boldsymbol{a}_j[w]$*, send an "abort" message to the ideal functionality and to* B^* *(on behalf of Alice) and terminate the simulation with the output of* B^**.*
   - *If* B^* *outputs an abort symbol, send an abort message to the ideal functionality, and terminate the simulation with the output of* B^**.*
6. *Send to* B^* *a random* $\Delta \leftarrow \mathbb{F}_k$*.*
7. *Receive* $\boldsymbol{\alpha}, \boldsymbol{h}$ *from* B^**, set*

$$\boldsymbol{v}_S := (\boldsymbol{v}_j)_{j \in S}, \quad \boldsymbol{a}_S := (\boldsymbol{a}_j)_{j \in S}, \quad \boldsymbol{a}' := \mathsf{Ext}(\boldsymbol{a}_S), \quad \boldsymbol{u}' := \mathsf{Ext}(\boldsymbol{v}_S) + \Delta \mathsf{Ext}(\boldsymbol{a}_S),$$

   send to the ideal VOLE functionality the vectors $\boldsymbol{f}' := \boldsymbol{\alpha} + \boldsymbol{a}'$ *and* $\boldsymbol{g}' := \boldsymbol{h} - \boldsymbol{u}'$*, and terminate with the output of* B^**.*

We analyze the simulator.

Lemma 7.9. *For every security parameter* k*, and inputs* $x \in \mathbb{F}_k$ *and* $\boldsymbol{f}, \boldsymbol{g} \in \mathbb{F}_k^W$ *for Alice and Bob respectively, Bob's simulated view in the ideal execution concatenated with Alice's output is statistically indistinguishable from the Bob's view and Alice's output in the real execution. Specifically, the statistical distance is at most* $1/|\mathbb{F}_k| = \mathsf{negl}(k)$*.*

Proof. Consider the real execution and the simulated execution (which appear in Table 3 for convenience). Fix the random tape of B^* and the inputs $x, \boldsymbol{f}, \boldsymbol{g}$, accordingly we can also treat $(M, (\boldsymbol{c}_j, I_j)_{j \in [t]}) := B^*(\boldsymbol{f}, \boldsymbol{g})$ and $\boldsymbol{a}_j \leftarrow \text{SIM}_1(M, \boldsymbol{c}_j, I_j)$ as fixed values in both experiments. Next, observe that the *simulated* random variables

$$(\Delta, (\boldsymbol{v}_j)_{j \in [t]})$$

are uniformly distributed and so are the *real-execution* random variables

$$(x - x^*, (x^* \boldsymbol{a}_j + \boldsymbol{b}_j)_{j\in[t]}),$$

as induced by the random choice of x^* and $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_t$. Let us condition on $\boldsymbol{v}_j = x^*\boldsymbol{a}_j + \boldsymbol{b}_j$ and on the event that $\Delta = x - x^*$. Furthermore, let us arbitrarily fix the value of $\boldsymbol{v}_j$ for every $j \in [t]$.

Since SIM perfectly simulates Π, the random variables $(\boldsymbol{z}_j[i])_{i\notin I_j}, (\boldsymbol{d}_j[i])_{i\in I_j}$, $\forall j \in [t]$ are identically distributed in both experiments. Let us condition on some fixing of these variables, and note that the view of B^* remains identical up to Step (5).

Next, we show that, except with probability $1/|\mathbb{F}_k|$ over the choice of x^* (or equivalently Δ), Step (5) aborts in the real execution if and only if it also aborts in simulated execution. First, assume that the simulation does not abort in Step (5). That is,

$$\lambda' = \sum_{j\in S} \boldsymbol{v}_j[w] \quad \text{and} \quad \delta = \sum_{j\in S} \boldsymbol{a}_j[w].$$

Plugging in $\boldsymbol{v}_j = x^*\boldsymbol{a}_j + \boldsymbol{b}_j$, we get that

$$\lambda' = x^* \sum_{j\in S} \boldsymbol{a}_j[w] + \sum_{j\in S} \boldsymbol{b}_j[w] = x^*\delta + \boldsymbol{b}_j[w],$$

and therefore Step (5) in the real execution does not abort as well (for any choice of x^*). For the other direction, observe that the simulation aborts if either (1) $\delta \neq \sum_{j\in S} \boldsymbol{a}_j[w]$ or (2) $\delta = \sum_{j\in S} \boldsymbol{a}_j[w]$ and $\lambda' \neq \sum_{j\in S} \boldsymbol{v}_j[w]$. In the latter case, $\lambda' \neq x^*\delta + \sum_{j\in S} \boldsymbol{b}_j[w]$ and so the real execution must abort. In the former case, the probability that B^* outputs

$$\lambda' = x^*\delta + \sum_{j\in S} \boldsymbol{b}_j[w] = x^* \left(\delta - \sum_{j\in S} \boldsymbol{a}_j[w]\right) + \sum_{j\in S} \boldsymbol{v}_j[w]$$

is $1/|\mathbb{F}_k|$. Indeed, since x^* is uniformly distributed and $\delta \neq \sum_{j\in S} \boldsymbol{a}_j[w]$, the RHS is uniform even conditioned on the value of $(\boldsymbol{v}_j)_{j\in[t]}$ and so on B^*'s view.

We continue the analysis and move on to Step 6 (assuming that both experiments did not abort before this point). Since $\Delta = x - x^*$, the view of B^* remains identical after Step 6 as well. By fixing $\Delta = x - x^*$ to some arbitrary value, we also get some fixed value for $\boldsymbol{\alpha}, \boldsymbol{h}$. Finally, by linearity, we can write Alice's output in the simulated distribution as

$$\begin{aligned} x(\boldsymbol{\alpha} + \boldsymbol{a}') + \boldsymbol{h} - \boldsymbol{u}' &= x\boldsymbol{\alpha} + x\mathsf{Ext}(\boldsymbol{a}_S) + \boldsymbol{h} - \mathsf{Ext}(\boldsymbol{v}_S) - \Delta\mathsf{Ext}(\boldsymbol{a}_S) \\ &= \boldsymbol{h} + x\boldsymbol{\alpha} + (x - \Delta)\mathsf{Ext}(\boldsymbol{a}_S) - \mathsf{Ext}(\boldsymbol{v}_S) \\ &= \boldsymbol{h} + x\boldsymbol{\alpha} + \mathsf{Ext}(x^*\boldsymbol{a}_S - \boldsymbol{v}_S) \\ &= \boldsymbol{h} + x\boldsymbol{\alpha} - \mathsf{Ext}(\boldsymbol{b}_S), \end{aligned}$$

which is equal to the output of Alice in the real execution. The lemma follows. □

#	Real execution	Simulated execution
1	$(M,(\boldsymbol{c}_j,I_j)_{j\in[t]}) := B^*(\boldsymbol{f},\boldsymbol{g})$	
2	Sample $x^* \leftarrow \mathbb{F}_t$ and $\boldsymbol{b}_j \leftarrow \mathbb{F}_k^w, \forall j\in[t]$	Sample $\boldsymbol{v}_j \leftarrow \mathbb{F}_k^w, \forall j\in[t]$ and $\Delta \leftarrow \mathbb{F}_k$
3	$\forall j\in[t]$ set $(\boldsymbol{z}_j[i])_{i\notin I_j}, (\boldsymbol{d}_j[i])_{i\in I_j} \leftarrow \Pi_A(M,x^*,\boldsymbol{b}_j,\boldsymbol{c}_j)$ Pass to B^*, $\forall j\in[t]$: $(\boldsymbol{z}_j[i])_{i\notin I_j}, (\boldsymbol{d}_j[i])_{i\in I_j}$	$\forall j\in[t]$ set $\boldsymbol{a}_j \leftarrow \textsc{Sim}_1(M,\boldsymbol{c}_j,I_j)$ and $(\boldsymbol{z}_j[i])_{i\notin I_j}, (\boldsymbol{d}_j[i])_{i\in I_j} \leftarrow \textsc{Sim}_2(M,\boldsymbol{c}_j,I_j,\boldsymbol{a}_j,\boldsymbol{v}_j)$
4	**Abort** if B^* aborts	
5	$(\delta,S) := B^*$ $\lambda' := B^*(\mathsf{commit})$ **Abort** if $\lambda' \neq x^*\delta + \sum_{j\in S}\boldsymbol{b}_j[w]$ $B^*(\mathsf{open},\lambda')$	**Abort** if $\lambda' \neq \sum_{j\in S}\boldsymbol{v}_j[w] \vee \delta \neq \sum_{j\in S}\boldsymbol{a}_j[w]$
6	$(\boldsymbol{\alpha},\boldsymbol{h}) := B^*(x-x^*)$	$(\boldsymbol{\alpha},\boldsymbol{h}) := B^*(\Delta)$
7	Set $\boldsymbol{b}_S := (\boldsymbol{b}_j)_{j\in S}$ $\boldsymbol{b}' := \mathsf{Ext}(\boldsymbol{b}_S)$ Output $\boldsymbol{h} + x\boldsymbol{\alpha} - \boldsymbol{b}'$ and B^*'s output	Set $\boldsymbol{a}_S := (\boldsymbol{a}_j)_{j\in S}$, $\boldsymbol{v}_S := (\boldsymbol{v}_j)_{j\in S}$ $\boldsymbol{a}' := \mathsf{Ext}(\boldsymbol{a}_S)$, $\boldsymbol{u}' := \mathsf{Ext}(\boldsymbol{v}_S) + \Delta\mathsf{Ext}(\boldsymbol{a}_S)$ Output $x(\boldsymbol{\alpha}+\boldsymbol{a}') + \boldsymbol{h} - \boldsymbol{u}'$ and B^*'s output

Table 3: The real execution and the simulated execution. Empty cells in the right column (simulation) indicate that the simulator applies the same operation that is applied in the real experiment as appeared in the corresponding cells in the left column. We implicitly assume that whenever an abort occurs, we output Bob's view and an abort symbol for Alice. We let Π_A denote Alice's computation in the protocol Π (Protocol 2), and let $\textsc{Sim}_1$ (resp., $\textsc{Sim}_2$) denote the first (resp., second) step of the simulator of Bob in Π (Simulator 1). That is $\textsc{Sim}_1$ extracts an effective input which is passed to $\textsc{Sim}_2$ together with the value $\boldsymbol{v}_j$ that is supposedly given to it by the ideal w-RVOLE functionality. To avoid cluttering, some intermediate variables were omitted and values were computed directly.

8 CDS for span membership

Our goal in this section is to construct an efficient CDS scheme for the span membership function. Recall that for a matrix $T \in \mathbb{F}^{m\times n}$ and a vector $\boldsymbol{c} \in \mathbb{F}^m$, the span membership predicate $f_{T,\boldsymbol{c}} : \{0,1\}^m \to \{0,1\}$ receives a vector $I \in \{0,1\}^m$ and accepts it if and only if $\boldsymbol{c}[I] \in \mathsf{colspan}\left(T[I]\right)$. Our CDS protocol $(\mathsf{Enc}_0,\mathsf{Enc}_1,\ldots,\mathsf{Enc}_m)$ is defined as follows.

Construction 8.1 (CDS for span membership)**.** *Given public parameters $T \in \mathbb{F}^{m\times n}$ and $\boldsymbol{c} \in \mathbb{F}^m$, secret $\zeta \in \mathbb{F}$ and random row vector $\boldsymbol{h} \in \mathbb{F}^m$ we define the CDS as follows. Let*

$$\mathsf{Enc}_0(\zeta;\boldsymbol{h}) := (\boldsymbol{h}T, \boldsymbol{h}\cdot\boldsymbol{c}+\zeta) \in \mathbb{F}^n \times \mathbb{F}$$

and, for $1 \le i \le m$, let

$$\mathsf{Enc}_i(I_i,\zeta;\boldsymbol{h}) := (1-I_i)h_i = \begin{cases} h_i & \text{if } I_i = 0 \\ 0 & \text{if } I_i = 1 \end{cases}.$$

Claim 8.2 (correctness)**.** *There exists a decoder algorithm* Dec *that perfectly decodes the CDS whenever I satisfies the predicate f. Furthermore, the algorithm can be implemented by making only $O(n+m+L)$ arithmetic operations where L is the number of operations needed in order to left-multiply a vector by the matrix T.*

Proof of claim. Given a vector I, an offline CDS message $(\boldsymbol{\gamma}, \alpha) \in \mathbb{F}^n \times \mathbb{F}$ (supposedly $\boldsymbol{\gamma} = \boldsymbol{h}T$ and $\alpha = \boldsymbol{h} \cdot \boldsymbol{c} + \zeta$), and a vector of CDS online messages $(\beta_i)_{1 \le i \le m} \in \mathbb{F}^m$, the decoder proceeds as follows. (1) The decoder aggregates the elements $(\beta_i)_{i \in [m]}$ into a row vector $\boldsymbol{\beta}$ of length m, and computes the vector

$$\boldsymbol{\varphi} := \boldsymbol{\gamma} - \boldsymbol{\beta} \cdot T = \boldsymbol{h}T - \boldsymbol{\beta} \cdot T = (\boldsymbol{h} - \boldsymbol{\beta})T = \boldsymbol{h}[I]T = \boldsymbol{h}T[I] \in \mathbb{F}^n$$

where the last equality follows by noting that the ith entry of $\boldsymbol{\beta}$ is h_i if $I_i = 0$ and zero if $I_i = 1$. Namely $\boldsymbol{\beta} = \boldsymbol{h}[\bar{I}]$. (2) Next the decoder computes the scalars

$$y := \boldsymbol{\beta} \cdot \boldsymbol{c} = (\boldsymbol{h} - \boldsymbol{h}[I]) \cdot \boldsymbol{c} = \boldsymbol{h} \cdot \boldsymbol{c} - \boldsymbol{h} \cdot \boldsymbol{c}[I]$$

and

$$t := \alpha - y = \boldsymbol{h} \cdot \boldsymbol{c} + \zeta - \boldsymbol{h} \cdot \boldsymbol{c} + \boldsymbol{h} \cdot \boldsymbol{c}[I] = \boldsymbol{h} \cdot \boldsymbol{c}[I] + \zeta.$$

(3) Finally, assuming that the input I satisfies the predicate, the decoder finds a vector $\boldsymbol{v} \in \mathbb{F}^n$ such that $\boldsymbol{c}[I] = T[I] \cdot \boldsymbol{v}$, computes the value

$$s := \boldsymbol{\varphi} \cdot \boldsymbol{v} = \boldsymbol{h}T[I] \cdot \boldsymbol{v} = \boldsymbol{h} \cdot \boldsymbol{c}[I]$$

and outputs the result

$$t - s = \boldsymbol{h} \cdot \boldsymbol{c}[I] + \zeta - \boldsymbol{h} \cdot \boldsymbol{c}[I] = \zeta$$

as required. The total number of arithmetic operations is $O(L)$ for Step (1), $O(m)$ for step (2), and $O(n)$ for step (3), as required. □

Claim 8.3 (privacy)**.** *Suppose that the predicate's evaluation on I gives 0, then the CDS messages perfectly hide the secret ζ.*

Proof of claim. Fix $\zeta, T, \boldsymbol{c}$ and I for which $\boldsymbol{c}[I] \notin \mathsf{colspan}\left(T[I]\right)$. Let $\boldsymbol{h}$ be a uniform vector. It suffices to show that conditioned on the online messages $\boldsymbol{h}[\bar{I}]$ and on the first part of the offline message $\boldsymbol{h}T$, the second part of the offline message $\boldsymbol{h}\boldsymbol{c} + \zeta$ is uniform. By simple linear algebra, to show this, it suffices to prove that $\boldsymbol{h}[I]\boldsymbol{c}[I]$ is uniformly distributed conditioned on $\boldsymbol{h}[I]T[I]$. Indeed, this follows immediately from the fact that $\boldsymbol{c}[I]$ is linearly independent of the columns of $T[I]$. The claim follows. □

Computational complexity. The offline server computes the left multiplication of the vector $\boldsymbol{h} \in \mathbb{F}^m$ to the matrix $T \in \mathbb{F}^{m \times n}$ and the inner product of $\boldsymbol{h}$ with $\boldsymbol{c}$, both can be done by a linear-size arithmetic circuit of size $O(m)$. While this is immediate for the inner-product operation, it also applies to the left multiplication operation since T is computable by a linear-size linear circuit and by exploiting the "generalized transposition principle" (see [Bor57],[IKOS08, Thm 3.2],[Dru13, Lemma 3.1.6]). The remaining servers run in a time of $O(1)$. The decoder also performs left vector-matrix multiplication and inner products in time of $O(m)$ and also additional computations in time of $O(n)$ (which is also $O(m)$ by our protocol setup).

Communication complexity. The offline server communicates $n+1$ field elements. The ith online server communicates a single field element if its input bit I_i equals to zero, and needs no communication if the bit is 1. The *online communication* in the former case can be further reduced to k bits (where k is the security parameter) at the expense of relaxing the security to computational via the following optimization. Say that a (truly) random field element can be sampled by using $t = \mathsf{poly}(k)$ random bits. Then, we let each server sample a short k-bit string s_i for a PRG $G : \{0,1\}^k \to \{0,1\}^t$, and set h_i to the pseudorandom field element that is sampled based on the string $G(s_i)$. When $I_i = 0$, the server sends the seed s_i, and when $I_i = 1$, the sender remains silent (or sends the default message 0). The decoder maps every received seed s_i to h_i, and proceeds as in the original protocol. Privacy follows along with the previous analysis while noticing that $\boldsymbol{h}[\bar{I}]$ is pseudorandom given the view of the receiver. The computational complexity is not affected assuming that G can be computed by making $O(t)$ binary operations. Such a generator can be obtained assuming the existence of a linear-stretch pseudorandom generator $G : \{0,1\}^k \to \{0,1\}^{2k}$ which is also computable in linear-time $O(k)$ which in turn follows from a binary variant of Assumption 4.1. Alternatively, one can completely remove the *offline* message at the expense of increasing the computational complexity. This can be done by sampling $\boldsymbol{h}$ conditioned on $\boldsymbol{h}T = 0^n$ and omitting the offline message (which is now fixed to the all-zero vector). Since the original protocol is perfectly private and perfectly correct, this variant is also perfectly private and perfectly correct. To implement this approach, we can prepare the matrix $H \in \mathbb{F}^{m-n \times m}$ that spans the co-kernel of T ahead of time (in a one-time preprocessing step when T is generated), and when the CDS is invoked, we can sample a vector $\boldsymbol{h}$ from the cokernel of T with a computational complexity of $O(m(m-n))$. In fact, this operation can be computed in a (non-reusable) offline phase before the input is known. Note that this optimization is not compatible with the previous one that selects each entry of $\boldsymbol{h}$ via a PRG. However, it preserves perfect privacy (and perfect correctness) and it can be applied even when the field is small.

Remark 7 (Validity Testing)**.** *Recall that for VOLE3, our CDS should be efficiently tested so that given a full vector of CDS messages, $\boldsymbol{z}$, one should be able to accept every honestly generated $\boldsymbol{z}$ and reject every* invalid *vector. (A vector $\boldsymbol{z}$ is valid if for every input I that satisfies the predicate the same secret is being recovered.) It is not hard to verify that our CDS (and all the above variants) admits such efficient validity testing. Specifically, given the offline CDS messages $(\boldsymbol{\gamma}, \alpha) \in \mathbb{F}^n \times \mathbb{F}$ and the vector $\boldsymbol{h} \in \mathbb{F}^m$ of online messages, the tester verifies that $\boldsymbol{h}T = \boldsymbol{\gamma}$.*

9 Batch-OLE

In this section, we use a polynomial-stretch arithmetic $\mathbf{NC^0}$ PRG $G : \mathbb{F}^k \to \mathbb{F}^n$ to reduce n copies of OLE (of width 1) to k copies of $O(n/k)$-width VOLE with constant computational overhead. Following [IKOS08, ADI+17] we rely on a combination of Beaver's OT extension [Bea96] with an arithmetic decomposable randomized encoding and show how to extend it to the active setting.

9.1 From vector-OLE to Generalized Affine Functionalities

Let $n(k)$ be a polynomial in k and let $G : \mathbb{F}^k \to \mathbb{F}^n$ be a function that can be computed by a constant-depth bounded fan-in arithmetic circuit (aka $\mathbf{NC^0}$ arithmetic circuit). We assume that G is *input-regular* in the sense that each input affects at most $O(n/k)$ outputs. Let $f_G : \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^n$

be the mapping

$$(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{x}) \mapsto G(\boldsymbol{x}) \odot \boldsymbol{c} + \boldsymbol{d},$$

where $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{F}^n$, $\boldsymbol{x} \in \mathbb{F}^k$ and $\odot$ stands for entry-wise multiplication.

The G-generalized affine functionality. We will be interested in computing the *Generalized Affine* Functionality $\mathcal{F}_G$ which takes the vectors $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{F}^n$ from Alice (the sender) and delivers to the receiver Bob a random vector $\boldsymbol{x} \in \mathbb{F}^k$ together with the outcome of $f_G(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{x})$. The functionality $\mathcal{F}_G$ is corruption-aware and it allows a malicious Bob to choose $\boldsymbol{x}$ arbitrarily. Throughout, we assume that, for every $i \in [n]$, the output of $G_i(\cdot)$ is a non-constant function (i.e., for some x, x', it holds that $G_i(x) = 0$ and $G_i(x') = 1$). This is without loss of generality since the task of securely computing such outputs is trivial (just let Alice send the value $\boldsymbol{c}_i \cdot G_i + \boldsymbol{d}_i$). We will also be interested in computing a generalization of $\mathcal{F}_G$ in which for an honest Bob, $\boldsymbol{x}$ is chosen from some (possibly) non-uniform distribution $X = (X_1, \ldots, X_k)$ over $\mathbb{F}^k$. We denote this variant by $\mathcal{F}_{G,X}$ and restrict our attention to the case where, for every $i \in [n]$, the marginal distribution

$$(X_j : j \in S_i), \quad \text{where } S_i \text{ is the set of inputs that influence the } i\text{th output of } G,$$

is uniform over $\mathbb{F}^{|S_i|}$. We refer to such distributions as being G-*compatible*. Clearly, the uniform distribution $U(\mathbb{F}^k)$ is G-compatible and indeed $\mathcal{F}_{G,U(\mathbb{F}^k)}$ is simply $\mathcal{F}_G$. Also note that for every G-compatible distribution X, it holds that $G_i(X) \equiv G_i(U(\mathbb{F}^k))$ for every $i \in [n]$.

Realizing $\mathcal{F}_{G,X}$. In the passive setting, it is easy to realize $\mathcal{F}_{G,X}$ by using an arithmetic variant of Yao's protocol [Yao86] where the garbled circuit is replaced with fully-decomposable randomized encoding. In fact, this protocol also provides active security against the receiver Bob. We will show how to cheaply upgrade the protocol and provide active security against the sender as well. Our protocol employs DARE for $f = f_G$. By Fact 3.2, there exists a DARE $\hat{f}$ which can be encoded and decoded by an $O(n)$-size arithmetic circuit. Since $\hat{f}$ is decomposable we can write it as

$$\hat{f}(\boldsymbol{x}, \boldsymbol{c}, \boldsymbol{d}; \boldsymbol{r}) = (\hat{f}_0(\boldsymbol{c}, \boldsymbol{d}; \boldsymbol{r}), (\hat{f}_i(x_i; \boldsymbol{r}))_{i \in [k]}).$$

(That is, we collapse the $\boldsymbol{c}$-dependent outputs and the $\boldsymbol{d}$-dependent outputs to a single "block".) Furthermore, for every $i \in [k]$ the function $\hat{f}_i(x_i; \boldsymbol{r})$ can be written as $x_i \boldsymbol{a}_i + \boldsymbol{b}_i$ where the vectors $(\boldsymbol{a}_i, \boldsymbol{b}_i)$ are sampled by a "key-sampling" mapping $K_i : \boldsymbol{r} \mapsto (\boldsymbol{a}_i, \boldsymbol{b}_i)$. By padding, we may assume that the key generation functions $K_1, \ldots, K_k$ have uniform output length of w, and since G is an input-regular $\mathbf{NC^0}$ function, it holds that $w = O(n/k)$. Moreover, recall that given $(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{d})$ we can collectively compute $\hat{f}_0(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{d}), (K_i(\boldsymbol{r}))_{i \in [k]}$ by $O(n)$ arithmetic operations. We denote the randomness complexity of the DARE by ρ. We realize $\mathcal{F}_{G,X}$ by Protocol 5 which employs an ideal batch-VOLE of width $2w$ and length k (that is, k parallel calls to VOLE of width $2w$), and performs 2 additional rounds of interaction. In Section 9.3 we prove the following lemma.

Lemma 9.1. *For every G-compatible distribution X, Protocol 5 realizes $\mathcal{F}_{G,X}$ with information-theoretic active security in a constant number of rounds by making a single call to an ideal k-length $O(n/k)$-width VOLE, communicating $O(n)$ field elements, and performing $O(n)$ arithmetic operations. The protocol is perfectly secure against an active adversary that corrupts Bob (receiver) and statistically secure against an adversary that actively corrupts Alice (the sender) where the statistical deviation is $O(D/|\mathbb{F}|) = \mathsf{negl}(k)$ where $D = O(1)$ is the degree of G.*

We note that the protocol securely computes $\mathcal{F}_{G,X}$ even if G is not in $\mathbf{NC^0}$, as long as the ratio, $D/|\mathbb{F}|$, between the degree D of G and the field size, is negligible. Of course, in this case, the computational complexity may be larger depending on the complexity of $\hat{f}$.

Protocol 5 (GA protocol). *Given an input $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{F}^n$ for Alice and an empty input for Bob, the protocol proceeds as follows.*

1. ***Alice:*** *selects randomness $\boldsymbol{r} \leftarrow \mathbb{F}^\rho$ for the encoding, and sets $\boldsymbol{v}_0 = \hat{f}_0(\boldsymbol{c}, \boldsymbol{d}; \boldsymbol{r})$ and $(\boldsymbol{a}_i, \boldsymbol{b}_i) = K_i(\boldsymbol{r})$ for every $i \in [k]$. In addition, Alice samples random $\boldsymbol{c}', \boldsymbol{d}' \in \mathbb{F}^n$ and $\boldsymbol{r}' \leftarrow \mathbb{F}^\rho$, and sets $\boldsymbol{v}'_0 = \hat{f}_0(\boldsymbol{c}', \boldsymbol{d}'; \boldsymbol{r})$ and $(\boldsymbol{a}'_i, \boldsymbol{b}'_i) = K_i(\boldsymbol{r}')$ for every $i \in [k]$.*
2. *The parties invoke k-length of $O(n/k)$-width VOLE as follows. Bob samples $\boldsymbol{x} \leftarrow X$, and plays the role of the receiver with the input $\boldsymbol{x} = (x_1, \ldots, x_k)$. Alice plays the role of the sender and sets her inputs to be the $2w$-length vectors $(\boldsymbol{a}_i \circ \boldsymbol{a}'_i)$ and $(\boldsymbol{b}_i \circ \boldsymbol{b}_i)$ for every $i \in [k]$ where $\circ$ denotes concatenation. For every $i \in [k]$, the functionality delivers to Bob the vectors $\boldsymbol{v}_i = x_i \boldsymbol{a}_i + \boldsymbol{b}_i$, $\boldsymbol{v}'_i = x_i \boldsymbol{a}'_i + \boldsymbol{b}'_i$.*
   In addition, Alice sends to Bob the vectors $\boldsymbol{v}_0$ and $\boldsymbol{v}'_0$.
3. ***Bob:*** *decodes the vectors $\boldsymbol{z}, \boldsymbol{z}' \in \mathbb{F}^n$ by setting $\boldsymbol{z} = \mathsf{Dec}((\boldsymbol{v}_i)_{0 \le i \le k})$ and $\boldsymbol{z}' = \mathsf{Dec}((\boldsymbol{v}'_i)_{0 \le i \le k})$ where Dec is the decoder of the DARE. In addition, Bob sends to Alice a random non-zero field element $\alpha \leftarrow \mathbb{F}^*$.*
4. ***Alice:*** *sends to Bob the n-length vectors $\boldsymbol{\gamma} = \boldsymbol{c} + \alpha \boldsymbol{c}'$ and $\boldsymbol{\delta} = \boldsymbol{d} + \alpha \boldsymbol{d}'$.*
5. ***Bob:*** *outputs $(\boldsymbol{x}, \boldsymbol{z})$ if $\boldsymbol{z} + \alpha \boldsymbol{z}' = G(\boldsymbol{x}) \odot \boldsymbol{\gamma} + \boldsymbol{\delta}$; Otherwise, Bob aborts with failure.*

9.2 From Generalized Affine Functionalities to batch-OLE

Following Beaver [Bea96], one can easily construct batch-OLE of length n based on a single call to the $\mathcal{F}_G$ functionality where G is a PRG (see also [ADI+17, Lemma 3]). This transformation naturally extends to the active setting and to the case where $\mathcal{F}_{G,X}$ is being employed, as long as the distribution $G(X)$ is pseudorandom. Formally,

Lemma 9.2. *Let G be an efficiently computable function and let X be an efficiently samplable probability distribution over $\mathbb{F}^k$ such that $G(X)$ is computationally indistinguishable from the uniform distribution over $\mathbb{F}^n$.*[25] *Then, there exists an actively-secure constant-round protocol that realizes batch-OLE of length n by making a single call to $\mathcal{F}_{G,X}$, communicating $O(n)$ field elements and performing $O(n)$ arithmetic operations. The protocol is perfectly secure against an actively-corrupt receiver and computationally secure against an actively-corrupt sender.*

Proof. Let $\boldsymbol{y} = (y_i)_{i \in [n]}$ be the input of Alice (the receiver) and let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$ be the input of Bob (the sender). Our goal is to pass the value $\boldsymbol{y} \odot \boldsymbol{a} + \boldsymbol{b}$ to Alice.

1. Alice and Bob call the functionality $\mathcal{F}_{G,X}$ where Bob's input is the vectors $\boldsymbol{a}, \boldsymbol{c} \in \mathbb{F}^n$ where $\boldsymbol{c} \leftarrow \mathbb{F}^n$ is chosen uniformly at random. If the functionality sends an abort symbol to some party, then this party aborts. Otherwise, the functionality delivers to Alice the vector $\boldsymbol{x} \in \mathbb{F}^k$ and the vector $\boldsymbol{u} = G(\boldsymbol{x}) \odot \boldsymbol{a} + \boldsymbol{c}$.

2. Alice sends to Bob the vector $\boldsymbol{\Delta} = \boldsymbol{y} - G(\boldsymbol{x})$.

[25] Formally, $\mathbb{F}, G$, and X are all an infinite ensemble of fields, functions, and distributions that are parameterized by the security parameter k.

3. Bob responds with $\boldsymbol{v} = \boldsymbol{\Delta} \odot \boldsymbol{a} + (\boldsymbol{b} - \boldsymbol{c})$.

4. Alice outputs $\boldsymbol{z} = \boldsymbol{u} + \boldsymbol{v}$.

We sketch the security proof starting with a corrupted Alice A^*. The simulation $\textsc{Sim}_{\mathrm{A}^*}(\boldsymbol{y})$, invokes Alice $\mathrm{A}^*(\boldsymbol{y})$ and gets an input $\boldsymbol{x}$ for $\mathcal{F}_{G,X}$ or an "abort" order for Bob. In the latter case, we ask the ideal batch-OLE functionality to abort Bob and output Alice's output. In the former case, the simulator feeds Alice with a random vector $\boldsymbol{u} \leftarrow \mathbb{F}^n$ as the answer of $\mathcal{F}_{G,X}$. Alice responds with a vector $\boldsymbol{\Delta}$, and the simulator calls the ideal batch-OLE functionality with $\boldsymbol{\Delta} + G(\boldsymbol{x})$ and receives a vector $\boldsymbol{z}$. The simulator passes to Alice the vector $\boldsymbol{v} = \boldsymbol{z} - \boldsymbol{u}$ and terminates with Alice's output.

Analysis: Fix Bob's input to $\boldsymbol{a}, \boldsymbol{b}$ and fix Alice's message to $\mathcal{F}_{G,X}$ (either an abort request of $\boldsymbol{x} \in \mathbb{F}^k$). Observe that the simulation aborts if and only if the real execution aborts and in this case the simulation is perfect. Let us assume that Alice does not issue an abort request. Then, in both experiments $\boldsymbol{u}$ is uniformly distributed, the vector $\boldsymbol{v}$ equals to $\boldsymbol{z} - \boldsymbol{u}$ and the vector $\boldsymbol{z}$ can be written as $(\boldsymbol{\Delta} + G(\boldsymbol{x})) \odot \boldsymbol{a} + \boldsymbol{b}$, thus the simulation perfectly emulates Alice's view, as required.

To simulate a corrupted Bob B^*. The simulation $\textsc{Sim}_{\mathrm{B}^*}(\boldsymbol{a}, \boldsymbol{b})$, invokes Bob $\mathrm{B}^*(\boldsymbol{a}, \boldsymbol{b})$ and gets an input $\boldsymbol{a}', \boldsymbol{c}$ for $\mathcal{F}_{G,X}$ or an "abort" order for Alice. In the latter case, we ask the ideal batch-OLE functionality to abort Alice and output Bob's output. In the former case, the simulator sends to Bob a random vector $\boldsymbol{\Delta} \leftarrow \mathbb{F}^n$ on behalf of Alice. Bob responds with the value $\boldsymbol{v}$ and the simulator calls the ideal functionality with the value $(\boldsymbol{a}', \boldsymbol{c} + \boldsymbol{v})$.

Analysis: Fix Alice's input $\boldsymbol{y}$, Bob's inputs $\boldsymbol{a}, \boldsymbol{b}$, and Bob's initial message $\boldsymbol{a}', \boldsymbol{c}$ or an "abort" instruction to $\mathcal{F}_{G,X}$. If Bob asks $\mathcal{F}_{G,X}$ to abort Alice then the output of the real experiment (i.e., the concatenated outputs of Alice and Bob) and the output of the simulated experiment are identically distributed. Let us condition on the event that Bob does not ask to abort Alice. We claim that the outputs of the two experiments are computationally indistinguishable. Consider a hybrid experiment that is identical to the simulator except that $\boldsymbol{\Delta} = \boldsymbol{y} - G(\boldsymbol{x})$ where $\boldsymbol{x} \leftarrow X$. Since $G(X)$ is pseudorandom the outcome of this experiment is computationally indistinguishable from the output of the simulated experiment. Also, observe that the outcome of the hybrid experiment is distributed identically to the real experiment. Indeed, Bob's view is identically distributed in both experiments, and Alice's output in the real experiment is

$$\boldsymbol{z} = \boldsymbol{u} + \boldsymbol{v} = G(\boldsymbol{x}) \odot \boldsymbol{a}' + \boldsymbol{c} + \boldsymbol{v},$$

which is exactly the same output that the ideal functionality delivers to her in the simulated experiment. The lemma follows. □

We can now prove Theorem 2.2, restated here for convenience:

Theorem 9.3 (Theorem 2.2 restated)**.** *Assuming the existence of an* $\mathbf{NC^0}$ *arithmetic PRG* $G : \mathbb{F}^k \to \mathbb{F}^n$ *where* $n = k^c$ *for* $c > 1$*, the batch-OLE functionality of length* n *can be realized with active security in the VOLE-hybrid model with arithmetic complexity of* $O(n)$ *and by making a single call to an ideal* k*-length* $O(n/k)$*-width VOLE where the batch-OLE receiver (resp., sender) plays the role of the VOLE receiver (resp., sender). The protocol is perfectly secure against an active adversary that corrupts the receiver and computationally secure against an adversary that actively corrupts the sender. Moreover, the protocol has a constant number of rounds.*

Proof. If $G : \mathbb{F}^k \to \mathbb{F}^n$ is input regular, the theorem follows directly from Lemmas 9.1 and 9.2. First, realize the batch OLE given a single call to $\mathcal{F}_G$ and then realize $\mathcal{F}_G$ with complexity $O(n)$ and a single call to an ideal k-length $O(n/k)$-width VOLE.

The input regularity requirement is satisfied by most natural candidates for $\mathbf{NC^0}$ PRG's. Still, we show that it can be completely waived as follows. Given an $\mathbf{NC^0}$ PRG G which is not input-regular, we can find every high-influence input that touches at least $t > 10n/k$ outputs and duplicate it $tk/10n$ times. As a result, we get an input regular $\mathbf{NC^0}$ mapping $G' : \mathbb{F}^{k'} \to \mathbb{F}^n$ where, by Markov's inequality, $k' = O(k)$. Now consider the probability distribution X that samples a "seed" $\boldsymbol{x}' \in \mathbb{F}^{k'}$ non-uniformly by sampling $\boldsymbol{x} \leftarrow \mathbb{F}^k$ and expanding it to $\boldsymbol{x}'$ by duplicating the entries that correspond to the duplicated inputs. Observe that $G'(X) \equiv G(U_k)$ is pseudorandom and so, by Lemma 9.2, the functionality $\mathcal{F}_{G',X}$ suffices for realizing batch-OLE with constant overhead. Moreover, X is G'-compatible since 2 copies of the same input are never connected to the same output, and so, by Lemma 9.1, the functionality $\mathcal{F}'_{G',X}$ reduces to a single call to an ideal k-length $O(n/k)$-width VOLE with $O(n)$ arithmetic complexity. The theorem follows. □

On the concrete complexity of the protocol. The complexity of the protocol is dominated by the parameters of the PRG G and the cost of the DARE for f_G. Assuming that $G : \mathbb{F}^k \to \mathbb{F}^n$ is a d-local regular PRG whose DARE can be computed and decoded by T arithmetic operations, Protocol 5 has a complexity of $2T$ plus $4n$ additions and $4n$ multiplications. The value of T depends on the locality d and the exact choice of the predicate that computes G_i, and deserves further study. Recall that Protocol 5 essentially realizes "pseudrandom" batch-OLE. This can be upgraded to a "standard" batch-OLE via Lemma 9.2, with an additional overhead of $2n$ multiplications and $5n$ additions.

9.3 Proof of Lemma 9.1

The complexity statement can be easily verified. We analyze the security of the protocol starting with the case of an actively corrupted Bob B^*. The simulator $\textsc{Sim}_{\mathrm{B}^*}$ operates as follows.

Simulator 6 (Bob's Simulator for $\mathcal{F}_{G,X}$). $\textsc{Sim}_{\mathrm{B}^*}$

1. *Call* B^* *and get* $\boldsymbol{x}$ *as the input to the ideal batch-VOLE oracle.*
2. *The simulator* B^* *calls the ideal functionality* $\mathcal{F}_{G,X}$ *with the input* $\boldsymbol{x}$ *for a corrupted receiver, and gets back the vector* $\boldsymbol{z} \in \mathbb{F}^n$*. The simulator calls the DARE simulator* $\textsc{Sim}(\boldsymbol{z})$ *to sample an encoding* $\hat{\boldsymbol{v}}$*. In addition, the simulator samples a vector* $\boldsymbol{z}' \leftarrow \mathbb{F}^n$ *and generates a corresponding encoding* $\hat{\boldsymbol{v}}' \leftarrow \textsc{Sim}(\boldsymbol{z}')$*. The encodings* $\hat{\boldsymbol{v}}$ *and* $\hat{\boldsymbol{v}}'$ *are given to* B^* *(parsed as the messages of Step 2 in the protocol).*
3. *Given a response* $\alpha \in \mathbb{F}^*$ *from* B^**, the simulator samples a vector* $\boldsymbol{\gamma} \leftarrow \mathbb{F}^n$*, sets* $\boldsymbol{\delta} = \boldsymbol{z} + \alpha\boldsymbol{z}' - G(\boldsymbol{x}) \odot \boldsymbol{\gamma}$*, and sends the pair* $(\boldsymbol{\gamma}, \boldsymbol{\delta})$ *to* B^**. We terminate the simulation with the final output of* B^**.*

We analyze the simulator. Fix an input $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{F}^n$ for Alice and fix $\boldsymbol{x}$ to be the first message of B^*. Note that this fixes the value $\boldsymbol{z} = G(\boldsymbol{x}) \odot \boldsymbol{c} + \boldsymbol{d}$ that the simulator receives from the ideal functionality. Let $\boldsymbol{c}', \boldsymbol{d}' \in \mathbb{F}^n$ be the vectors that are uniformly sampled in the real execution and observe that this means that the random variable $f_G(\boldsymbol{c}', \boldsymbol{d}', \boldsymbol{x}) = G(\boldsymbol{x}) \odot \boldsymbol{c}' + \boldsymbol{d}'$ is also uniform. Let us condition on the event that the vector $\boldsymbol{z}' \leftarrow \mathbb{F}^n$, which is uniformly sampled by the simulator, equals to $f_G(\boldsymbol{c}', \boldsymbol{d}', \boldsymbol{x})$. Then, by the perfect privacy of the DARE, Bob's view after the first round of interaction (Step 2)

is distributed identically in both experiments. Fix the first round messages (and the value of $\boldsymbol{z}'$). It suffices to show that, conditioned on this fixing, the vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ are distributed identically in both experiments. Focusing on the real execution, observe that, conditioned on $\boldsymbol{z}'$, the vector $\boldsymbol{c}'$ is uniform and $\boldsymbol{d}' = \boldsymbol{z}' - G(\boldsymbol{x}) \odot \boldsymbol{c}'$. Therefore, in the real execution, the vector $\boldsymbol{\gamma} = \boldsymbol{c} + \alpha \boldsymbol{c}'$ is uniform, and

$$\boldsymbol{\delta} = \boldsymbol{d} + \alpha \boldsymbol{d}' = (\boldsymbol{z} - G(\boldsymbol{x}) \odot \boldsymbol{c}) + \alpha(\boldsymbol{z}' - G(\boldsymbol{x}) \odot \boldsymbol{c}') = \boldsymbol{z} + \alpha \boldsymbol{z}' - G(\boldsymbol{x}) \odot (\boldsymbol{c} + \alpha \boldsymbol{c}') = \boldsymbol{z} + \alpha \boldsymbol{z}' - G(\boldsymbol{x}) \odot \boldsymbol{\gamma},$$

which is exactly the same distribution as in the simulated execution.

We move on to handle the case of a corrupted Alice $\mathrm{A}^*(\boldsymbol{c}^*, \boldsymbol{d}^*)$. The simulator $\textsc{Sim}_{\mathrm{A}^*}(\boldsymbol{c}^*, \boldsymbol{d}^*)$ operates as follows.

Simulator 7 (Alice's Simulator for $\mathcal{F}_{G,X}$). $\textsc{Sim}_{\mathrm{A}^*}(x)$*:*

1. *Call* $\mathrm{A}^*(\boldsymbol{c}^*, \boldsymbol{d}^*)$ *and get the VOLE inputs* $(\boldsymbol{a}_i \circ \boldsymbol{a}'_i, \boldsymbol{b}_i \circ \boldsymbol{b}'_i)_{i \in [k]}$ *and the messages* $(\boldsymbol{v}_0, \boldsymbol{v}'_0)$.
2. *For every* $i \in [n]$ *the simulator defines the circuits* $Q_i(\boldsymbol{x}) = \mathsf{Dec}_i(\boldsymbol{v}_0, x_i \cdot \boldsymbol{a}_i + \boldsymbol{b}_i)$ *and* $Q'_i(\boldsymbol{x}) = \mathsf{Dec}_i(\boldsymbol{v}'_0, x_i \cdot \boldsymbol{a}'_i + \boldsymbol{b}'_i)$ *where* Dec_i *computes the* i*th output bit of the DARE decoder* Dec*. Observe that* Q_i *and* Q'_i *are degree-*D *multivariate polynomials over* $\mathbb{F}$*. The simulator now checks if the polynomials* Q_i *and* Q'_i *are spanned by the polynomials* $\{G_i, 1\}$ *where* 1 *is the constant-*1 *polynomial. If the test passes let us record the coefficients* c_i, d_i, c'_i, d'_i *for which* $Q_i = c_i G_i + d_i$ *and* $Q'_i = c'_i G_i + d'_i$*. Otherwise, raise an internal "abort" flag.*
3. *The simulator samples a random non-zero scalar* $\alpha \leftarrow \mathbb{F}^*$ *and passes it to Alice who responds with the vectors* $\boldsymbol{\gamma}, \boldsymbol{\delta} \in \mathbb{F}^n$*.*
4. *If an abort flag was raised, send an abort symbol to the ideal functionality* $\mathcal{F}_{G,X}$ *on behalf of Alice and terminate with the output of* A^**.*
5. *Define the vectors* $\boldsymbol{c} = (c_i)_{i \in [n]}$*,* $\boldsymbol{d} = (d_i)_{i \in [n]}$*,* $\boldsymbol{c}' = (c'_i)_{i \in [n]}$ *and* $\boldsymbol{d}' = (d'_i)_{i \in [n]}$*. If* $\boldsymbol{\gamma} = \boldsymbol{c} + \alpha \boldsymbol{c}'$ *and* $\boldsymbol{\delta} = \boldsymbol{d} + \alpha \boldsymbol{d}'$*, call the ideal* $\mathcal{F}_{G,X}$ *with the vectors* $\boldsymbol{c}$ *and* $\boldsymbol{d}$ *as the inputs of the corrupted receiver; Otherwise, send the ideal functionality an abort symbol.*

Before analyzing the simulator observe that since Q_i and Q'_i are polynomials of constant degree D one can efficiently check if they are spanned by $\{G_i, 1\}$ and can efficiently recover the corresponding coefficients in case they exist.[26]

We analyze the simulator. Let us condition on the event that in both experiments the same non-zero scalar α is being sent to Alice. Since Alice's view consists of α, it suffices to analyze the output of Bob. First, observe that if the simulator does not abort, the simulation is perfect. Indeed, letting $\boldsymbol{c}, \boldsymbol{c}', \boldsymbol{d}$ and $\boldsymbol{d}'$ be the vectors that are computed in the simulation, we can write the vectors $\boldsymbol{z}$ and $\boldsymbol{z}'$ that are decoded by Bob in the real execution as

$$\boldsymbol{z} = \boldsymbol{c} \odot G(\boldsymbol{x}) + \boldsymbol{d}, \quad \text{and} \quad \boldsymbol{z}' = \boldsymbol{c}' \odot G(\boldsymbol{x}) + \boldsymbol{d}',$$

[26] In fact, even if the degree D is super-constant we can check this efficiently with a one-sided error of $O(\frac{D}{|\mathbb{F}|})$ that can be amplified to $(\frac{D}{|\mathbb{F}|})^t$ by making t repetitions. Indeed, sample 3 random values $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_2$ and check if there exists a solution c_i, d_i to the linear system $Q_i(p_j) = c_i G_i(p_j) + d_i$ for $j = 1, 2, 3$. The algorithm outputs a correct answer unless we hit a root of Q_i which happens with probability at most $O(\frac{D}{|\mathbb{F}|})$.

where $\boldsymbol{x} \leftarrow X$. Furthermore, $\boldsymbol{\gamma} = \boldsymbol{c} + \alpha \boldsymbol{c}'$ and $\boldsymbol{\delta} = \boldsymbol{d} + \alpha \boldsymbol{d}'$ (since the simulator does not abort) and so, in the real execution, the condition $\boldsymbol{z} + \alpha \boldsymbol{z}' = G(\boldsymbol{x}) \odot \boldsymbol{\gamma} + \boldsymbol{\delta}$ holds which means that Bob does not abort, and outputs the pair $(\boldsymbol{x}, f_G(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{x}))$, just like in the simulation.

Next, we claim that if the simulator raises an abort flag in Step 2, then, except with a probability of $(1+D)/|\mathbb{F}|$ over the choice of $\boldsymbol{x} \leftarrow X$ and $\alpha \leftarrow \mathbb{F}^*$, Bob also outputs an abort symbol in the real execution. Fix some i for which Q_i is not spanned by $\{G_i, 1\}$. (The case where Q'_i is not spanned is symmetric since $\alpha \neq 0$ is invertible.) We will need the following simple claim.

Claim 9.4. *If Q_i is not spanned by $\{G_i, 1\}$, then* $\Pr_\alpha[Q_i + \alpha Q'_i \in \mathrm{span}(G_i, 1)] \leq \frac{1}{|\mathbb{F}|-1}$.

Proof of Claim. Case 1: Suppose that Q'_i is in the span of $\{G_i, 1\}$. Then, for *every* α, $Q_i + \alpha Q'_i \notin \mathrm{span}(G_i, 1)$. Indeed, if both $Q_i + \alpha Q'_i$ and Q'_i are in the span of $\{G_i, 1\}$ then so is Q_i, in contradiction to our assumption. Case 2: Suppose that Q'_i is not in the span of $\{G_i, 1\}$. Then $\left\{Q'_i, G_i, 1\right\}$ are linearly independent and so there exists at most a single scalar $t \in \mathbb{F}$ for which $Q_i \in tQ'_i + \mathrm{span}(G_i, 1)$. Hence,

$$\Pr_\alpha[Q_i + \alpha Q'_i \in \mathrm{span}(G_i, 1)] = \Pr_\alpha[\alpha = -t] \leq \frac{1}{|\mathbb{F}^*|},$$

and the claim follows. □

Since α is chosen independently of Q_i and Q'_i, except with probability $1/(|\mathbb{F}| - 1)$, it holds that $Q_i + \alpha Q'_i \notin \mathrm{span}(G_i, 1)$. Let us condition on this event, and note that this means that the polynomial $Q_i(\cdot) + \alpha Q'_i(\cdot)$ is not equal to the polynomial $\gamma_i G_i(\cdot) + \delta_i$. Also note that, since our DARE is obtained by concatenating DAREs for each output of f_G, the polynomials Q_i and Q'_i depend only on $\boldsymbol{x}[S_i]$, the entries of $\boldsymbol{x}$ that affect the ith output of f_G. That is, $Q_i, Q'_i : \mathbb{F}^{S_i} \to \mathbb{F}$. By the Schwartz–Zippel lemma [Zip79, Sch80], it holds that

$$\Pr_{\boldsymbol{x} \leftarrow X}[Q_i(\boldsymbol{x}[S_i]) + \alpha Q'_i(\boldsymbol{x}[S_i]) = \gamma_i G_i(\boldsymbol{x}[S_i]) + \delta_i] \leq D/|\mathbb{F}|,$$

where we make use of the fact that X is G-compatible (i.e., the input $X[S_i]$ to Q_i and to Q'_i is distributed uniformly), and that both the LHS polynomial and the RHS polynomial are of degree D. Recalling that in the real execution $z_i = Q_i(\boldsymbol{x}[S_i])$, $z'_i = Q'_i(\boldsymbol{x}[S_i])$ for a randomly chosen $\boldsymbol{x} \leftarrow X$ (distributed independently of Q_i, Q'_i and α), we conclude that Bob's test in the real execution $z_i + \alpha z'_i = G_i(\boldsymbol{x}[S_i]) \cdot \gamma_i + \delta_i$ fails except with probability $D/|\mathbb{F}|$. Overall, by a union bound, Bob aborts except with probability $\frac{1}{|\mathbb{F}|-1} + \frac{D}{|\mathbb{F}|}$.

Finally, assume that the simulator does not raise an abort flag in Step 2 but aborts in Step 5. Then, for some $i \in [n]$, it holds that $\gamma_i \neq c_i + \alpha c'_i$ or $\delta_i \neq d_i + d'_i$ but the polynomials Q_i and Q'_i equal to the polynomials $c_i G_i + d_i$ and $c'_i G_i + d'_i$ respectively. Since G_i is non-constant, we have

$$Q_i + \alpha Q'_i \equiv (c_i + c'_i) G_i + d_i + d'_i \not\equiv \gamma_i G_i + \delta_i,$$

where $\equiv$ and $\not\equiv$ stand for polynomial equality and inequality, respectively. Again, by the Schwartz–Zippel lemma [Zip79, Sch80], the probability, over a random $\boldsymbol{x} \leftarrow X$, that

$$(c_i + c'_i) G_i(\boldsymbol{x}[S_i]) + d_i + d'_i = \gamma_i G_i(\boldsymbol{x}[S_i]) + \delta_i$$

is at most $D/|\mathbb{F}|$. It follows that in the real experiment, Bob aborts except with probability $D/|\mathbb{F}|$. Overall, the statistical deviation is at most $\max(\frac{1}{|\mathbb{F}|-1} + \frac{D}{|\mathbb{F}|}, \frac{D}{|\mathbb{F}|})$. Lemma 9.1 follows. □

10 Concrete Efficiency and Implementation

In this section, we analyze the concrete efficiency of the VOLE1 protocol (obtained by applying the RVOLE-to-VOLE transformation to Protocol 2) and present an implementation in Python. Our analysis and implementation can be used as a core for the VOLE2 protocol as well. The concrete computational complexity is essentially the same and the communication overhead per VOLE entry is minor (a single field element). To make the result as platform-independent as possible, we mainly focus on concrete arithmetic complexity which is a machine-independent measure. In Section 10.4 we provide some timing and profiling results for our concrete implementation over a specific (modest) machine. Our implementation can be found in [Kon22].

10.1 Concrete Computational Complexity

Reminder: main parameters. Recall that M is a $m \times k$ matrix that is partitioned to a top $u \times k$ part, M_{top}, and a bottom $v \times k$ part M_{bot} where $m = u + v$. (That is, M expands a k-long seed to a m-long vector whose μ-noisy version is pseudorandom). We also employ an error-correcting code C that maps w-long information words to v-long codewords where the width of the VOLE is w. We will measure the computational complexity (resp., communication complexity) **cost** as a function of VOLE width w and will be especially interested in the amortized cost per entry, i.e., **cost**$/w$. To this end, let us denote the blow-up factor of the error correcting code C by $\beta = v/w > 1$ and let $\alpha = u/w$. The constant $\beta > 1$ will always be taken to be a relatively small constant whereas $\alpha < 1$ is typically tiny since w is about 2 orders of magnitude larger (asymptotically, $\alpha = o(1)$). Note that $k < u = \alpha w$ and that $m = (v + u) \leq (\beta + \alpha)w \approx \beta w$.

Online/Offline realization. Through this section, we assume that the public parameters, which consist of the pseudorandom matrix M and the generating matrix C of the error-correcting code, are pre-computed and loaded to memory upon initialization. In addition, we split the computation in the protocol into an *offline phase* which is input-independent and so it can be executed in advance and a relatively light-weight *online phase* which depends on the input. We emphasize that the offline phase is communication-free and can be computed locally based on public parameters and private randomness. As a result, Alice (resp., Bob) can precompute this step even without knowing who will be the partner in the VOLE computation. For a matrix/linear operator K, we let E_K denote the complexity of computing a vector in the image of K. We assume that this cost is *uniform* over the rows in the sense that for a matrix K' obtained by selecting each row of K with probability α, it holds that the expected value of $E_{K'}$ is αE_K. This assumption holds for all our candidate distributions for both M and C.[27] We let $D_{K,\mu}$ denote the complexity of decoding a "codeword" of K under random erasures where the erasure probability is μ. (we omit μ when it is clear from the context)It will be convenient to further split $D_{K,\mu}$ to an offline part $D_{K,\mu,\mathsf{off}}$ and an online part $D_{K,\mu,\mathsf{on}}$ where the offline part computes some advice based on a given (randomly chosen) erasure pattern (and on K) and where the online part decodes the received codeword under the given erasure pattern based on the pre-computed advice. Most of the complexity will be counted as part of encoding/decoding procedures over M and C. We refer to all other operations as "external". As we will see, the cost of external operations will be marginal. The VOLE1 protocol can be described as follows.

[27] This assumption is not necessary and is only taken in order to obtain tighter concrete bounds.

B0: Offline Bob: prepares the pseudorandom noisy codeword $\boldsymbol{c}' = M\boldsymbol{r} + \boldsymbol{e} \in \mathbb{F}^m$ by sampling $\boldsymbol{r} \leftarrow \mathbb{F}^k$ and $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F})^m_\mu$. Let $I = I(e)$ and let $I_{\text{top}} \in \{0,1\}^u$ denote the vector the contains the first u entries of I. Bob also pre-computes a decoding advise for $M_{\text{top}}[I_{\text{top}}]$ and advice for the code C based on the erasure pattern induced by e. The overall complexity of this step is $E_M + D_{M_{\text{top}},\mu,\mathsf{off}} + D_{C,\mu,\mathsf{off}}$ plus an external (expected) cost of $\mu m = \mu(\beta+\alpha)w$ additions.[28]

A0: Offline Alice: pre-computes some of the CDS encoding $\boldsymbol{\gamma} = \boldsymbol{h}T$ by randomly sampling a row vector $\boldsymbol{h} \leftarrow \mathbb{F}^m$. Alice also prepares the vector $\boldsymbol{d}' = M\boldsymbol{r}' + 0^u \circ \mathsf{Ecc}(\boldsymbol{b}')$ by sampling $\boldsymbol{r}' \leftarrow \mathbb{F}^k$, $\boldsymbol{b}' \leftarrow \mathbb{F}^w$ and $x' \leftarrow \mathbb{F}$. Recall that the cost of left multiplication by the matrix T is the same as the cost of right multiplication in T (by the "transposition principle") which is $E_M + E_C$. Hence, the overall cost of this step is $2(E_M + E_C)$ plus an external cost of $v = \beta w$ additions.

In the online phase:

B1: Bob encodes his input $\boldsymbol{a}$ via the code C, adds it to the bottom part of the pseudorandom vector $\boldsymbol{c}'$, and sends the result $\boldsymbol{c}$ to Alice. The complexity is E_C plus an external cost of $v = \beta w$ additions.

A1: Alice samples a scalar x' sets $\Delta = x - x'$ and computes the vector $\boldsymbol{d} := x'\boldsymbol{c} + \boldsymbol{d}'$. Alice also computes the online part of the CDS by computing the inner-product $\boldsymbol{h}\cdot\boldsymbol{c}+\Delta$. The (external) complexity of this step is at most $2m + 1 = 2(\beta+\alpha)w + 1$ additions and multiplications.

OT: The parties call the batch-OT channel.

B2: Bob:

- Applies the CDS-decoder and recovers Δ. Decoding consists of left-multiplication by $T\left[\overline{I}\right]$ which on expectation[29] costs $\mu(E_M+E_C)$ plus an external cost of $\mu\cdot m+2(k+w) < w(\mu(\alpha+\beta)+2\alpha+2)$ additions and $\mu\cdot m+k+w < w(\mu(\alpha+\beta)+\alpha+1)$ multiplications.
- Solves the affine system defined by the "top" parts of $M[I]$ and the top part of the vector $\boldsymbol{d}[I]$ and recovers a solution $\boldsymbol{s}$. This is done in time $D_{M_{\text{top}},\mu,\mathsf{on}}$ based on the pre-computed advice.
- Computes the vector $\boldsymbol{d}'' = M_{\text{bot}}[I_{\text{bot}}] \cdot \boldsymbol{s}$ in expected time of $(1-\mu)E_{M_{\text{bot}}}$.
- Computes $\boldsymbol{d}_{\text{bot}}[I_{\text{bot}}] - \boldsymbol{d}''$, Applies the C-decoder to the vector $\boldsymbol{d}_{\text{bot}}[I_{\text{bot}}] - \boldsymbol{d}''$, shifts the decoded vector, $\boldsymbol{v} \in \mathbb{F}^w$, by $\Delta\boldsymbol{a} + \boldsymbol{b}$ and sends the output $\boldsymbol{u}$ to Alice. The complexity of this step is $D_{C,\mu,\mathsf{on}}$ plus external expected complexity of $(1-\mu)v+2w = w((1-\mu)\beta+2)$ additions and w multiplications.

A2: Alice subtracts $\boldsymbol{u} - \boldsymbol{b}'$ and terminates. The external complexity is w additions.

Overall the external operations accumulate to $w\left(\alpha(4+2\mu)+\beta(5+\mu)+5\right)+1$ addition and

$$w\left(\alpha(3+\mu)+\beta(2+\mu)+2\right)+1$$

multiplications. Ignoring the marginal αw terms, we get an amortized external complexity of $\beta(5+\mu)+5$ additions and $\beta(2+\mu)+2$ multiplications per VOLE entry. The overall costs are summarized in Table 4.

[28] The expectation is taken over the random choice of the noise vector since it is chosen in an iid form. By a Chernoff bound, the number of operations is concentrated around the expectation with a deviation of $o(m)$.

[29] The expectation is taken over the choice of the error pattern. Here we also use the "uniform cost" property of C and M.

Party	Offline	Online	Total
Alice	$2(E_M + E_C)$ $+\beta w$ add's	$(2\beta+1)w$ add's and $2\beta w$ mul's	$2(E_M + E_C)$ $+(3\beta+1)w$ add's and $2\beta w$ mul's
Bob	E_M $+D_{M_{\text{top}},\mu,\mathsf{off}} + D_{C,\mu,\mathsf{off}}$ $+\ \mu\beta w$ add's	$(1+\mu)E_C + \mu E_M + (1-\mu)E_{M_{\text{bot}}}$ $+D_{M_{\text{top}},\mu,\mathsf{on}} + D_{C,\mu,\mathsf{on}}$ $+(2\beta+4)w$ add's and $(2+\mu\beta)w$ mul's	$(1+\mu)(E_M + E_C) + (1-\mu)E_{M_{\text{bot}}}$ $+D_{M_{\text{top}},\mu} + D_{C,\mu}$ $+\ ((2+\mu)\beta+4)w$ add's and $(\mu\beta+2)w$ mul's
Total	$3E_M + 2E_C$ $+D_{M_{\text{top}},\mu,\mathsf{off}} + D_{C,\mu,\mathsf{off}}$ $+(1+\mu)\beta w$ add's	$\mu E_M + (1+\mu)E_C + (1-\mu)E_{M_{\text{bot}}}$ $+D_{M_{\text{top}},\mu,\mathsf{on}} + D_{C,\mu,\mathsf{on}}$ $+\ (4\beta+5)w$ add's and $((2+\mu)\beta+2)w$ mul's	$(3+\mu)(E_M + E_C) + (1-\mu)E_{M_{\text{bot}}}$ $+D_{M_{\text{top}},\mu} + D_{C,\mu}$ $+((5+\mu)\beta+5)w$ add's and $((2+\mu)\beta+2)w$ mul's

Table 4: Concrete efficiency

Some comments are in place:

1. **The CDS overhead compared to plain ADINZ:** The overhead of the protocol compared to plain ADINZ protocol consists of the CDS encoding (preprocessing and online) and decoding whose total complexity is dominated by $(1+\mu)(E_M + E_C)$. Since the total complexity of the ADINZ protocol is larger than $(3+\mu)(E_M + E_C)$, the additional overhead is at most 0.4 more expensive for $\mu < 0.5$. (In fact, it is typically much smaller since the overall cost is dominated by the decoding complexities $D_{M_{\text{top}},\mu,\mathsf{on}} + D_{C,\mu,\mathsf{on}}$ which remain unchanged in the new protocol.)

2. **OT cost:** The protocol consumes m batch-OT calls in which Alice is always the sender and Bob is always the receiver where $m = (\alpha+\beta)w$ (i.e., about β calls per VOLE entry). For now, we treat this as a unit-cost operation. As we will see in the next subsection this cost is indeed marginal in practice.

3. **Fast Receiver:** The computational complexity of the protocol is asymmetric. In particular, Alice, the VOLE receiver only performs two encoding $2(E_M + E_C)$ and does not decode at all. Moreover, these operations can be pre-computed, leading to an extremely fast online complexity of 2β additions/multiplication and about β OT-calls per VOLE entry. (Jumping ahead, in our concrete instantiation β is about 3.) As mentioned in the introduction this can be useful in some scenarios.

4. **Pipelining the decoding:** As we will later see, the computationally-heaviest part of the protocol is the offline decoding of M_{top}. It is useful to note that this part of the offline computation is only employed later in the protocol after the OT is completed and it does not affect Bob's first message. Consequently, even if one does not use an offline/online architecture, one can begin this computation concurrently with the first round and to the OT round. Thus, when the network is slow, this part of the computation does not affect the overall latency.

10.2 Concrete Building Blocks

To allow a fair comparison with [ADI+17] we use the same building blocks (pseudorandom matrix distribution, noise rate, error correcting code) as [ADI+17]. We briefly review these choices and note that a better choice of the ingredients may lead to better performance. The task of fine-tuning the parameters is left for future work.

10.2.1 Choice of the Matrix M and noise rate μ

In order to implement protocol 2 for VOLE, a fast pseudorandom matrix M is needed (see assumption 4.1). Following Applebaum et al. [ADI+17], we have chosen to use a random d-sparse matrix, namely, a random matrix that has exactly d non-zero random field elements in each row. This choice is inspired by previous works in the binary domain [Ale11, ABW10], and is analyzed concretely in [Zic17]. Specifically, we set the sparsity parameter d to 10, set the length parameter $v = k^2$ and $u = \lceil 1.4k \rceil$ so that the number of rows in the matrix is $m = u + v = 1.4k + k^2$, and take the noise rate, μ, to $1/4$.[30] The analysis in [Zic17, ADI+17] suggests that for $k = 182$ and $k = 240$ this suffices for security levels of approximately 80 and 100 bits of security respectively. (See Zichron's Master thesis [Zic17] and Section 6.1 in [ADI+17] for a more detailed overview of known attacks).

10.2.2 Complexity of encoding and erasure-decoding of M

Clearly, E_M costs dm additions and multiplications which for $d = 10$ equals to $10m$ leading to an amortized cost (per VOLE entry) of about $d\beta$. (Recall that β is the expansion factor of the error-correcting code C whose value will be determined in the next paragraph.) The erasure-decoding procedure is the most expensive step in our implementation and so we exploit the *online/offline* model as follows. In the *offline* phase, given $M_{\text{top}}[I_{\text{top}}] \in \mathbb{F}^{u \times k}$, we compute its lower–upper (LU) decomposition. That is, we compute a lower trapezoidal matrix $L \in \mathbb{F}^{u \times k}$ with a unit diagonal, an upper triangular matrix $U \in \mathbb{F}^{k \times k}$ with no zero elements in the diagonal, a row permutation matrix $P \in \mathbb{F}^{u \times u}$ (represented as a u sized vector over $[u]$) and a column permutation matrix $Q \in \mathbb{F}^{k \times k}$ (represented as a vector as well), for which $L \cdot U = P \cdot M_{\text{top}}[I_{\text{top}}] \cdot Q$. The matrices L, U, P and Q are left as "advice" for the online phase. In the *online* decoding, given $\boldsymbol{d}_{\text{top}}[I_{\text{top}}]$, all we need to do is to solve the linear system $L \cdot \boldsymbol{y} = P \cdot \boldsymbol{d}_{\text{top}}[I_{\text{top}}]$ for $\boldsymbol{y} \in \mathbb{F}^k$ via forward substitution (using only the top k rows of L), then solve $U \cdot \boldsymbol{z} = \boldsymbol{y}$ for $\boldsymbol{z} \in \mathbb{F}^k$ via backward substitution and finally compute $\boldsymbol{s} = Q \cdot \boldsymbol{z}$. Both first computations cost $k^2/2$ additions and multiplications and the multiplication by Q costs no arithmetic operations (since Q is a permutation matrix that has a single 1 element in each row and column). And so, the overall cost $D_{M_{\text{top}},\mu,\mathsf{on}}$, is k^2 additions and multiplications leading to an amortized overhead of β additions and multiplications per VOLE entry.

Realizing the offline phase. A naive approach for deriving an LU-decomposition (the "advice") is to rely on a Gaussian elimination algorithm. In expectation the cost $D_{M_{\text{top}},\mu,\mathsf{off}}$ is $0.35k^3$ additions and multiplications, leading to an amortized cost of $0.35k\beta$ additions and multiplications per VOLE

[30] Recall that in the theoretical sections, we suggested taking m to $O(k^3)$ in order to obtain a constant asymptotic complexity per VOLE element. The implementation of [ADI+17] avoids this optimization and takes $m = O(k^2)$ since the treatment of very long vectors slows down things due to memory management. We follow this choice and note that it significantly increases the amortized cost, especially for $D_{M_{\text{top}}}$.

entry.[31] However, it turns out that one can do much better by exploiting the fact that the matrix M is *sparse*. Specifically, we have implemented in Python the GPLU algorithm for sparse LU decomposition (introduced in [GP88], also described in [BD16]). The overall arithmetic complexity of the GPLU algorithm is proportional to the total number of multiplications in the computation of $L \cdot U$, denoted by flops(LU). Furthermore, this complexity also dominates all other operations performed by the algorithm. While we do not have a theoretical analysis of the flops(LU) for the case of random sparse matrices, our experiments show that this algorithm performs significantly better than the naive Gaussian-based approach. Specifically, for $k = 182$ (resp., $k = 240$) we get an amortized cost of 58 (resp., 78) arithmetic operations – more than 3-times better compared to the naive Gaussian Elimination approach. See Table 5 for accurate numbers. For completeness, we provide a high-level description of the GPLU algorithm.

The GPLU algorithm. Denoting the matrix $M_{\text{top}}[I_{\text{top}}]$ by $A \in \mathbb{F}^{u \times k}$ and its ith row by A_i (L_i and U_i are defined correspondingly). During the ith step of the algorithm, L_i and U_i are computed from A_i and the (already computed) previous $i-1$ rows of U as follows. By aggregating the unknown elements from L_i and U_i into a k sized vector $\boldsymbol{x}_i = (\ell_{i,1}, ..., \ell_{i,i-1}, u_{i,i}, ..., u_{i,k})$, the ith step of the decomposition is done by solving the triangular linear system $\boldsymbol{x}_i \cdot U'_i = A_i$ for $\boldsymbol{x}_i \in \mathbb{F}^k$ where $U'_i \in \mathbb{F}^{k \times k}$ is an upper-triangular matrix consists of the previous $i-1$ rows of U and additional $k-(i-1)$ ones on the diagonal. Such a triangular system can be solved sequentially from left to right (L_i to U_i) for each element in $\boldsymbol{x}_i$ at a time according to the following expressions:

$$\ell_{i,j} = \Big(A_{i,j} - \sum_{t=1}^{j-1} \ell_{i,t} U_{t,j}\Big) U_{j,j}^{-1} \quad \text{for } j \in [i-1] \quad \text{and} \quad u_{i,j} = A_{i,j} - \sum_{t=1}^{i-1} \ell_{i,t} U_{t,j} \quad \text{for } j \geq i.$$

Before solving the ith system, the GPLU algorithm exploits the sparsity of A to efficiently determine the sparsity pattern (non-zero entries) of $\boldsymbol{x}_i$. This phase of the algorithm is known as "Symbolic Decomposition". Using this technique, the algorithm can solve the triangular system only for the non-zero coefficients in $\boldsymbol{x}_i$ (and use only non-zero coefficients of L_i in the expressions above). As described in [GP88], the sparsity pattern of $\boldsymbol{x}_i$ can be found without performing any kind of numerical computation. Let $G_i = G(U'_i)$ be a directed graph over vertices from $[k]$ and with edges according to the adjacency matrix U'_i (where non-zero entries considered as 1). By the theorem of Gilbert and Peierls [GP88], the system can have a non-zero solution for entry $t \in [k]$ in $\boldsymbol{x}_i$ iff there exists a directed path in G_i between nodes s and t for some $s \in [k]$ such that $A_{i,s} \neq 0$. In other words, the sparsity pattern of $\boldsymbol{x}_i$ can be obtained by defining the sparsity pattern of A_i: $\mathcal{A}_i = \{j \in [k] | A_{i,j} \neq 0\}$ and finding (by depth-first search) all the reachable nodes in G_i from any node in $\mathcal{A}_i$. In order to perform the *online* decoding it suffices to compute the matrix U and only k rows of L, and so we perform the whole LU decomposition by solving k triangular linear systems. According to our setup, at each step, A_i is a sparse row with d non-zeros entries, and so $\mathcal{A}_i$ is a set of constant size d (which we set to be 10). The complexity analysis appears in [GP88]. Some comments regarding the GPLU algorithm are in place.

1. **The structure of the graph and partial pivoting:** since U'_i is triangular, each node $j \in [k]$ in $G_i = G(U'_i)$ can have directed edges only toward nodes of larger indices, or themselves (self-loop). In fact, in the ith step nodes of indices larger than $i-1$ can have only self-loops in

[31] More generally, letting $\gamma k > k$ denote the number of non-zero rows in $M_{\text{top}}[I_{\text{top}}] \in \mathbb{F}^{u \times k}$, the complexity is $\sum_{i=1}^{k} (\gamma k - i)(k - i) \leq k^3(\gamma - \frac{\gamma+1}{2} + 1/3)$ additions and multiplications. Since the expected value of γ is $(1-\mu)1.4k = 1.05$, in expectation the cost $D_{M_{\text{top}},\mu,\text{off}}$ is $0.35k^3$ additions and multiplications.

G_i (due to the diagonal extension with ones) and no further nodes are reachable from them. We can exploit these properties as follows. Before executing the GPLU algorithm, we select rows from A that have different positions of left-most coefficient (different pivot's positions) and put them on top of all other rows, sorted by increasing positions of pivots. Say that we select q rows, during any ith step of the decomposition for $i \leq q$, nodes of indices lower than i will be unreachable from $\mathcal{A}_i$ in G_i. As a result, the coefficients in L_i will all be zero (except for the ith coefficient of one) and the q selected rows will be copied as-is into U, without any depth search for a reachable node in G_i. We verified experimentally that, for our setup, about 25% of the rows in $M_{\text{top}}[I_{\text{top}}]$ can be copied directly into U by this technique.

2. **Maintaining the pivots:** For the k triangular systems to have a unique solution, we need to make sure that the diagonal coefficients of U'_i at each step are all non-zero. However, while solving the ith system we may find that $u_{i,i} = 0$. In such case, we search for a non-zero coefficient $u_{i,j}$ where $j > i$ and swap the columns j and i in U and (in next steps, if needed) also in A. If no non-zero coefficient can be found then the current row A_i is a linear combination of the previous rows, and so we drop it and choose another non-empty row in its place. (We can always find an alternative row assuming that A has a full rank of k).

10.2.3 ECC: Using Luby Transform Codes

Next, we choose an error correcting code that maps vectors of length w to codewords of length v and supports "fast" encoding and decoding under random erasures with erasure probability of 1/4. Ideally, we would like to minimize the code's overhead $\beta = v/w$ (i.e., maximize the parameter w). Again we follow [ADI+17] and use a fixed-block version of Luby Transform (LT) codes [Lub02].[32] The resulting code (ensemble) has a simple 0/1 generating matrix C with an average sparsity (number of 1's per row) of $s < 8$. As a result, the encoding algorithm performs only $E_C < sv = (s\beta)w$ additions (and no multiplications). We use "peeling"-based decoding whose cost is also at most $vs = (s\beta)w$ additions. This simplicity come at the expense of a sub-optimal blow-up factor of $\beta = v/w$ of ≈ 3.31 for $k = 182$ and ≈ 2.88 for $k = 240$, and a small decoding error of ≈ 0.01. (The error is calculated experimentally in [ADI+17].) We mention that the error depends only on the choice of the LT coding matrix and on the noise pattern (both selected by Bob) and so it cannot be influenced by the adversary and it is independent of the VOLE inputs. The error can be completely removed as explained in [ADI+17] at the expense of slightly strengthening the pseudorandomness assumption. A summary of the parameters is given in Table 6.

We can again exploit the *online/offline* model in order to precompute the list of *online* arithmetic operations needed for decoding. This can be done by running the peeling-based algorithm in a "symbolic" way. That is, in the *offline* stage, we scan the constraint graph (after erasures) and record a sequence of peeling operations that will be executed in the online phase. While the saving here is less dramatic than in the decoding process of M, our experiments show that this optimization allows us to push about 80%-fraction of the decoding work to the offline stage.

[32] Specifically, we sample a random generating matrix with rate 1/3 where each row is sampled independently from the *Robust Soliton distribution* with parameters $\delta = 0.01$ and $c = 1.17224$ for $k = 182$ and $c = 1.23075$ for $k = 240$.

10.2.4 Realizing the Oblivious Transfers

We realize the batch-OT via the actively-secure OT-extension of [KOS15] which operates in a constant number of rounds, has a constant amortized computational cost per OT (2 calls to a hash function and 2 operations over k-bit fields) and has a $o(1)$ communication overhead on top of the OT message lengths (increases the total communication by $O(k)$ bits independent of the number of OTs being performed). We use the C++ implementation of [Rin] and integrate it into our Python implementation via Python extension for C++ [(20]. The execution of this batch-OT implementation incurs some non-negligible overhead in setup time that is partially devoted to setting up a network channel between the parties. Later in Section 10.4, we present an experimental measurement of this setup time.

10.2.5 Concrete Arithmetic Complexity for concrete building blocks: Summary

We summarize the amortized arithmetic complexity per VOLE entry of each building block in Table 5. (We slightly abuse notation and treat E_M, $D_{M,\mathsf{off}}$,$D_{M,\mathsf{on}}$,E_C, D_C as amortized quantities, i.e., divide them by the VOLE width parameter w.) We then use the formulas from Table 4 to calculate the Offline, Online, and Total amortized arithmetic complexity of both parties in Table 6. To avoid clutter we do not distinguish between additions and multiplications and always use the maximal value between the two. In particular, the values form an upper bound on the number of multiplications. Following [BCGI18], we mention that Emmart et al. [ELWW16] report 12.2 billion modular multiplications per second over a field $GF(p)$ for a 128-bit prime p using a common graphics card (Nvidia GTX 980 Ti).

k	u	d	v	w	β	s	E_M	$D_{M,\mathsf{off}}$	$D_{M,\mathsf{on}}$	E_C	D_C
182	255	10	33124	10000	3.3124	6.718	33.124	57.5389	3.3124	22.252	3.359
240	336	10	57600	20000	2.88	7.7197	28.8	77.9686	2.88	22.232	3.85985

Table 5: Implementation parameters and amortized concrete arithmetic complexities. The pseudo-random matrix M is a random d-sparse $m \times k$ matrix. The error correcting code C is based on LT codes, it maps w vectors to $v = \beta w$-long codewords and has an average sparsity of s. We take $k = 182$ (resp., $k = 240$) to obtain a security level of approximately 80 (resp., 100) bits of security respectively. For some of the quantities, we have analytical expressions and some of them were analyzed experimentally. Specifically, recall that $E_M \approx d\beta$ additions and multiplications, $D_{M,\mathsf{off}}$ was experimentally measured, $D_{M,\mathsf{on}} \approx \beta$ additions and multiplications, $E_C \approx s\beta$ additions, and D_C was experimentally measured. (The latter quantity outperforms the naive theoretical upper bound of $s\beta$ which is based on the average column weight in C.)

	$k = 182$, $w = 10000$			$k = 240$, $w = 20000$		
Party	Offline	Online	Total	Offline	Online	Total
Alice	114.0658	7.6248	121.6906	104.9454	6.76	111.7054
Bob	91.491	78.236	169.727	107.4886	73.0907	180.5793
Total	205.5568	85.8608	291.4176	212.434	79.8507	292.2847

Table 6: Amortized arithmetic complexity for each party in our concrete implementation.

10.3 Communication Complexity

Given the above choices, we compute the concrete communication complexity of our implementation. Note that this calculation holds for an arbitrary field. (Indeed the noise rate μ, the sparsity parameter d, the matrix dimension m, and the code choice are all field-independent.) Recall that RVOLE1 communicates $2m + (k + w + 1) + w$ field elements (as explained in Remark 4 after Lemma 5.1). Recalling that $m = v + u = \beta w + 1.4k$, we get that the protocol communicates $(2\beta + 2)w + 3.8k + 1$ field elements. Since $k = \sqrt{\beta w}$, the communication rate of the VOLE is about $1/(2\beta + 2)$ which is approximately 1/8. (A better rate can be obtained by using a better error-correcting code and/or by using a smaller noise rate.)

10.4 Test Set-up and Results

Our setup consists of a machine with 8GB RAM and a 64-bit i7-8550U CPU running at 1.8GHz. A b-bit field is instantiated by choosing $\mathbb{F}_p$ for the largest prime $p < 2^b$. (The prime p was found by using standard prime tables). We benchmark our implementation with b-bit field for $b \in \{16, 32, 64, 128\}$. We present the amortized execution times (divided by w) of Protocol 2 (for both choices of parameters) in the last rows of Tables 7 and 8. We obtain these times by interactively executing the protocol 10000 times in a loop and counting only the successful execution. We partition the subroutines into offline procedures and online procedures, and for each subroutine (e.g., multiplication by M) we allocate a row and calculate its *fractional cost* in percentages by taking the ratio between the time that is devoted to the sub-procedure (including multiple calls if there are any). The total fractions that are devoted to the offline and online parts are given in the bottom rows just before the "total time" row. The setup time of the batch-OT implementation was experimentally calculated by executing a single OT instance for 10000 repetitions. We found that the average OT setup time is approximately $270.418ms$, and counted it as part of the offline phase.

Overall, the total amortized time complexity of Alice ranges from $32\mu s$ to $60\mu s$ (depending on the field size and on the VOLE width and security level), Bob's timing is typically slower by a factor of about 2, and the total amortized timing of Alice and Bob together ranges between $60\mu s$ to $140\mu s$.

Let us take a closer look at Alice's performance (Table 7). As predicted by the theoretical analysis, Alice's online computation, which is almost exclusively devoted to the OT, is relatively lightweight and consists of $12\% - 21\%$ of her total complexity. In the offline phase, Alice's work is also dominated by the setup complexity of the OT. Overall, the batch-OT (online and offline together) consumes more than 90% of Alice's time for small fields and more than 40% for her time for large fields. This is somewhat expected due to the use of relatively small-width VOLE which employs batch-OT over relatively few (w =10K,20K) instances. Larger values of w (for which batch-OT becomes extremely faster) are expected to lead to significant improvements in Alice's amortized complexity. For the current values of w, better batch-OT with a smaller amortization point can lead to major improvements. We also note that, for large fields, the cost of the offline-CDS, and the computation of M also take a noticeable fraction of Alice's total time (about 20% each at the extreme).

Moving to Bob (Table 8), we see that the offline phase is significantly heavier than the online phase (typically over 80%) and is mainly devoted to the offline OT setup and to the LU decomposition. Again offline OT setup becomes cheaper when w grows, and so for larger values, the LU-decomposition is expected to dominate the cost. Finally, Bob's online computation is domi-

nated by the online-OT part, though for large fields the computation of $M_{\text{bot}}[I_{\text{bot}}]s$ is almost as heavy.

On implementing finite fields arithmetics. Elements of large fields (with 128-bit elements) are represented by *large integers*, and field arithmetic (additions, multiplications) are realized by using Python's standard operators over large integers while modular reductions are applied as late as possible. For relatively small fields (16 bit and 32 bit) we achieve better performance using Python's native tools for linear algebraic operations and for sparse matrix operations. Specifically, we employ SciPy,a NumPy based library for scientific computations that supports operations over 64-bit integers. By embedding 16-bit fields over such integers we can emulate many (about 2^{32}) additions of quadratic terms without worrying about an overflow. We can therefore realize sparse matrix multiplications, either by M or by C (or by sub-matrices of them) via the library's fast algorithms for sparse matrix-vector multiplications. This optimization can be carried to 32-bit fields as follows. Firstly, we have enough magnitude to support native *addition* of many (say 2^{32}) elements without an overflow. This suffices for multiplying a vector by a sparse 0-1 matrix (since this involves only additions) like the Luby-transform matrix C. Secondly, in order to compute a fast vector multiplication by M (over a 32-bit field), we split each field element into two blocks and reduce the computation to 2 fast SciPy sparse matrix-vector multiplications over 16-bit integers each. The end results are merged back at the end to a single 64-bit integer using the Distributive Law and modular reduction. A similar decomposition is also employed for multiplying vectors by the 0-1 matrix C over 64-bit fields. These optimizations result in a very efficient realization of protocol 2 over relatively small fields, as presented later in Tables 7 to 8.

	$k = 182$, $w = 10000$				$k = 240$, $w = 20000$			
Field size	16 bit	32 bit	64 bit	128 bit	16 bit	32 bit	64 bit	128 bit
Linear map M	0.096%	4.083%	16.814%	16.355%	0.257%	6.958%	19.443%	19.642%
LT encoding	0.031%	0.418%	2.752%	5.133%	0.078%	0.482%	4.247%	6.992%
Offline CDS	0.173%	4.334%	16.895%	18.578%	0.567%	6.959%	20.448%	22.705%
OT setup	82.997%	71.909%	48.341%	44.935%	81.915%	64.333%	34.969%	30.142%
Other offline op's	0.082%	0.355%	2.347%	2.494%	0.066%	0.102%	2.95%	1.569%
OT Sender	16.414%	14.679%	9.92%	9.33%	16.85%	14.727%	12.169%	13.97%
Other online op's	0.207%	4.222%	2.93%	3.175%	0.267%	6.439%	5.774%	4.98%
Offline phase	83.379%	81.099%	87.149%	87.495%	82.883%	78.834%	82.057%	81.05%
Online phase	16.621%	18.901%	12.851%	12.505%	17.117%	21.166%	17.943%	18.95%
Total	32.581μs	37.605μs	55.939μs	60.178μs	16.505μs	21.017μs	38.664μs	44.857μs

Table 7: Alice's amortized benchmark.

References

[(20] Python Software Foundation (2022). Extending python with c or c++. `https://docs.python.org/3/extending/extending.html`.

[AAB17] Benny Applebaum, Jonathan Avron, and Chris Brzuska. Arithmetic cryptography. *J. ACM*, 64(2):10:1–10:74, 2017.

	$k = 182$, $w = 10000$				$k = 240$, $w = 20000$			
Field size	16 bit	32 bit	64 bit	128 bit	16 bit	32 bit	64 bit	128 bit
Linear map M	0.052%	2.188%	10.568%	10.04%	0.085%	2.597%	9.247%	9.159%
Offline LU	28.898%	30.635%	25.499%	25.529%	41.262%	37.677%	26.695%	26.902%
Offline LT decoding	11.363%	9.858%	7.78%	7.106%	19.715%	18.044%	12.468%	10.833%
OT setup	45.25%	38.527%	30.384%	27.586%	27.268%	24.016%	16.632%	14.055%
Other offline op's	0.012%	0.136%	0.475%	0.934%	0.01%	0.416%	0.9%	0.893%
LT encoding	0.017%	0.224%	1.729%	3.151%	0.026%	0.18%	2.02%	3.26%
OT Receiver	10.75%	9.958%	8.136%	7.989%	6.582%	6.776%	9.116%	9.387%
Online CDS	0.662%	2.597%	4.771%	4.381%	0.33%	3.061%	6.021%	5.731%
Online LU	0.949%	0.986%	1.455%	1.737%	1.387%	1.64%	1.86%	3.157%
Linear Map $M_{\text{bot}}[I_{\text{bot}}]$	0.047%	1.946%	5.627%	6.492%	0.067%	2.038%	8.932%	8.508%
Online LT decoding	1.934%	1.824%	1.451%	1.366%	2.992%	3.218%	2.85%	2.389%
Other online op's	0.066%	1.121%	2.125%	3.689%	0.276%	0.337%	3.259%	5.726%
Offline phase	85.575%	81.344%	74.706%	71.195%	88.34%	82.75%	65.942%	61.842%
Online phase	14.425%	18.656%	25.294%	28.805%	11.66%	17.25%	34.058%	38.158%
Total	59.76μs	70.188μs	88.997μs	98.025μs	49.584μs	56.297μs	81.289μs	96.199μs

Table 8: Bob's amortized benchmark.

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In *STOC*, pages 171–180, 2010.

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO*, pages 223–254, 2017.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in NC^0. *Comput. Complex.*, 17(1):38–69, 2008.

[AIK14] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. *SIAM J. Comput.*, 43(2):905–929, 2014.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *EUROCRYPT*, pages 119–135, 2001.

[AK19] Benny Applebaum and Eliran Kachlon. Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. In *FOCS*, pages 171–179, 2019.

[AL16] Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In *STOC*, pages 1087–1100, 2016.

[Ale11] Michael Alekhnovich. More on average case vs approximation complexity. *Comput. Complex.*, 20(4):755–786, 2011.

[ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *EUROCRYPT*, pages 673–701, 2015.

[App16] Benny Applebaum. Cryptographic hardness of random local functions - survey. *Computational Complexity*, 25(3):667–722, 2016.

[BCG+17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *ASIACRYPT*, pages 336–365, 2017.

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

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, pages 489–518, 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *CCS*, pages 896–912, 2018.

[BD16] Charles Bouillaguet and Claire Delaplace. Sparse gaussian elimination modulo p: An update. In *CASC*, pages 101–116, 2016.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In Gary L. Miller, editor, *STOC*, pages 479–488, 1996.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993.

[BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In *CRYPTO*, pages 92–122, 2021.

[Bor57] Jan Lourens Bordewijk. Inter-reciprocity applied to electrical networks. *Applied Scientific Research, Section A*, 6(1):1–74, 1957.

[CDI+19] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO*, pages 462–488, 2019.

[CGH+85] Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem of t-resilient functions (preliminary version). In *FOCS*, pages 396–407, 1985.

[CK88] Claude Crépeau and Joe Kilian. Weakening security assumptions and oblivious transfer (abstract). In *CRYPTO*, pages 2–7, 1988.

[Cré87] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In *CRYPTO*, pages 350–354, 1987.

[DI14] Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *ITCS*, pages 169–182, 2014.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In Stefano Tessaro, editor, *ITC 2021*, volume 199 of *LIPIcs*, pages 5:1–5:24. Schloss Dagstuhl, 2021.

[Dru13] Erez Druk. Linear time encodable codes and cryptography. Master's thesis, Technion, 2013.

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In *CRYPTO*, pages 205–210, 1982.

[ELWW16] Niall Emmart, Justin Luitjens, Charles C. Weems, and Cliff Woolley. Optimizing modular multiplication for nvidia's maxwell gpus. In *ARITH*, pages 47–54, 2016.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

[GI05] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Trans. Inf. Theory*, 51(10):3393–3400, 2005.

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000.

[GIKW14] Juan A. Garay, Yuval Ishai, Ranjit Kumaresan, and Hoeteck Wee. On the complexity of UC commitments. In *EUROCRYPT*, pages 677–694, 2014.

[Gil99] Niv Gilboa. Two party RSA key generation. In *CRYPTO*, pages 116–129, 1999.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC*, pages 495–504, 2014.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[GN17] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. *IACR Cryptol. ePrint Arch.*, page 1064, 2017.

[GNN17] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. *IACR Cryptol. ePrint Arch.*, page 409, 2017.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[GP88] John R. Gilbert and Tim Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. on Scientific and Statistical Comp.*, 9(5):862–874, 1988.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS*, pages 174–184, 1997.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, 2000.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *STOC*, pages 433–442, 2008.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, pages 294–314, 2009.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO*, pages 54–70, 2013.

[KLR10] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, 2010.

[Kon22] Niv Konstantini. Actively-secure-vector-ole. `https://github.com/NivKonst/Actively-Secure-Vector-OLE`, 2022.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *CRYPTO*, pages 724–741, 2015.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403, 2003.

[Lub02] Michael Luby. LT codes. In *FOCS*, page 271, 2002.

[MW08] Payman Mohassel and Enav Weinreb. Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries. In *CRYPTO*, pages 481–496, 2008.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, pages 245–254, 1999.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, page 187, 2005.

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

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. In *CCS*, pages 1055–1072, 2019.

[SSR08] Bhavani Shankar, Kannan Srinathan, and C. Pandu Rangan. Alternative protocols for generalized oblivious transfer. In *ICDCN*, pages 304–309, 2008.

[Tas11] Tamir Tassa. Generalized oblivious transfer by secret sharing. *Des. Codes Cryptogr.*, 58(1):11–21, 2011.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *SP*, pages 1074–1091, 2021.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[Zic17] Lior Zichron. Locally computable arithmetic pseudorandom generators. Master's thesis, Tel Aviv University, 2017.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

A Security of ADINZ against a malicious Alice

As discussed in Section 4.2, we can prove that the original RVOLE protocol of [ADI$^+$17] is secure against an active Alice under a new, arguably, natural intractability assumption. We first have to slightly modify the parameters of the ADINZ encoding matrix. Recall that the ADINZ encoding matrix T is defined as follows:

$$T = \left(\begin{array}{c|c} M_{m\times k} & \mathbf{0}_{u\times w} \\ & \mathsf{Ecc}_{v\times w} \end{array} \right),$$

where $m = \Omega(k^3)$, $u = \Omega(k \log^2 k)$ and $v = O(m)$. For technical reasons we will need to to slightly strengthen the linear-independence requirements of the matrix T as follows. Except with negligible probability over the choice of M and Ecc it must hold that: (a) If we sample a random subset of the first u rows of M by taking each row independently with probability $1 - 2\mu$ then the resulting matrix has full rank (all the columns are linearly independent); (b) The error-correcting code Ecc can recover from a constant fraction of arbitrary erasures which is slightly larger than 2μ. We introduce the following assumption:

Assumption A.1. *Let $S(T, \boldsymbol{u}, \boldsymbol{d}, I)$ be an indicator function that outputs 1 if $\boldsymbol{d}[I] \in \mathsf{colspan}\left((T|\boldsymbol{u})[I]\right)$ and zero otherwise.*

For every efficient adversary A^*, it holds that

$$(M, \boldsymbol{u}, \boldsymbol{d} \leftarrow \mathrm{A}^*(M, \boldsymbol{u}), S(T, \boldsymbol{u}, \boldsymbol{d}, I)) \quad \approx_c \quad (M, \boldsymbol{u}, \boldsymbol{d} \leftarrow \mathrm{A}^*(M, \boldsymbol{u}), S(T, \boldsymbol{u}, \boldsymbol{d}, I')), \tag{7}$$

where $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$, T is defined based on M as in Eq. (1),

$$\boldsymbol{u} = M\boldsymbol{r} + \boldsymbol{e} \qquad \text{for } \boldsymbol{r} \leftarrow \mathbb{F}_k^k, \boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m,$$

and $I = I(\boldsymbol{e})$ and $I' \leftarrow \mathsf{BER}^m(1 - \mu)$.

Intuitively the assumption asserts that, given a noisy codeword $\boldsymbol{u} = M\boldsymbol{r} + \boldsymbol{e}$ of M, it is hard to generate a new noisy codeword $\boldsymbol{d} = T\boldsymbol{r'} + \boldsymbol{e'}$ with respect to T whose noise $\boldsymbol{e'}$ is non-trivially correlated with the noise vector $\boldsymbol{e}$. Non-triviality here means that when the two vectors are projected to the set of clean coordinates $I = I(\boldsymbol{e})$ their agreement significantly deviates from their agreement when projected to a random sparse set I'.

Remark 8 (Game-based version of Assumption A.1)**.** *One can equivalently phrase the assumption via the following game between an adversary A^* and a Challenger:*

1. *The Challenger samples $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$ and $\boldsymbol{r} \leftarrow \mathbb{F}_k^k$, $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$. She sends to the adversary the matrix M and the noisy codeword $\boldsymbol{u} = M\boldsymbol{r} + \boldsymbol{e}$.*
2. *The adversary A^* responds with some vector $\boldsymbol{d}$.*
3. *The challenger tosses a coin $q \leftarrow \{0, 1\}$ and sends to A^* the value $S(T, \boldsymbol{u}, \boldsymbol{d}, I_q)$ where $I_0 = I(\boldsymbol{e})$ and $I_1 \leftarrow \mathsf{BER}^m(1 - \mu)$.*
4. *The adversary A^* sends a bit $q' \in \{0, 1\}$ and wins if $q = q'$.*

Assumption A.1 asserts that the winning probability of every efficient adversary is at most $0.5 + \mathsf{negl}(\kappa)$.

Lemma A.2. *Under Assumption A.1, the ADINZ protocol realizes the RVOLE functionality with active security against a malicious Alice.*

Proof. Fix some malicious Alice A, we define a simulator $\textsc{Sim}(x, \boldsymbol{b})$ as follows.

1. Sample $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$ and a uniform $\boldsymbol{c}' \leftarrow \mathbb{F}_k^m$ and sample randomly $I' \leftarrow \mathsf{BER}^m(1-\mu)$.
2. Call $\mathrm{A}(x, \boldsymbol{b})$ on the message $M, \boldsymbol{c}'$ and get back the vector $\boldsymbol{d} \in \mathbb{F}_k^m$ as the input that A sends to the m-batch OT oracle.
3. If $\boldsymbol{d}[I'] \notin \mathsf{colspan}\left((T|\boldsymbol{c}')[I']\right)$ ask the ideal functionality to deliver to Bob an "abort" message.
4. Else find a solution $(x', \boldsymbol{b}', \boldsymbol{r}')$ to the system of equations

$$\boldsymbol{d}[I'] = T[I'] \begin{pmatrix} \boldsymbol{r}' \\ \boldsymbol{b}' \end{pmatrix} + x' \boldsymbol{c}'[I'],$$

and send $(x', \boldsymbol{b}')$ to the ideal functionality as Alice's input.

We claim that the simulation is indistinguishable from the real interaction. For simplicity, and without loss of generality, let us assume that A is deterministic and so, apart of its inputs, its view consists of the values $M, \boldsymbol{c}$. Fix a sequence of inputs $\boldsymbol{a} = \{\boldsymbol{a}_k\}_{k\in\mathbb{N}}$ for Bob and a sequence of inputs $x = \{x_k\}_{k\in\mathbb{N}}, \boldsymbol{b} = \{\boldsymbol{b}_k\}_{k\in\mathbb{N}}$, we will show that the ensemble

$$(M, \boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a_k}) + \boldsymbol{e}, \boldsymbol{y}) \tag{8}$$

where $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$, $\boldsymbol{r} \leftarrow \mathbb{F}_k^k$, $\boldsymbol{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$ and $\boldsymbol{y}$ is the output of Bob in the protocol, is computationally indistinguishable from the ensemble simulated distribution

$$(M, \boldsymbol{c}' \leftarrow \mathbb{F}_k^m, \boldsymbol{y}') \tag{9}$$

where $\boldsymbol{y}'$ is the output of Bob in the simulated experiment. We prove this by considering the following hybrid experiments.

Hybrid H_1. Let H_1 be the experiment in which the simulator generates $\boldsymbol{c}$ by $E_{\boldsymbol{r}}(\boldsymbol{a_k}) + \boldsymbol{e}$ and other than that operates exactly as before. By the pseudorandomness assumption (Assumption 4.1 property 2), the tuple that is generated by this experiment:

$$(M, \boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a_k}) + \boldsymbol{e}, \boldsymbol{y}') \tag{10}$$

is computationally indistinguishable from the simulated tuple (9).

Hybrid H_2. The hybrid H_2 is defined similarly to H_1 except that in Step 3, the span-condition is being checked with respect to $I = I(\boldsymbol{e})$ where $\boldsymbol{e}$ is the same noise vector that is being used to generate $\boldsymbol{c}$. (In Step 4 we still use the random I'.)

Claim A.3. *The outcome of H_2, denoted by $(M, \boldsymbol{c}, \boldsymbol{y}'')$, is computationally indistinguishable from the outcome of H_1.*

Proof of Claim. We show that the modification in Step 3 is unlikely to change the abort decision. That is, it suffices to show that

$$(M, \boldsymbol{c}, \boldsymbol{d} = \mathrm{A}(M, \boldsymbol{c}), S(T, \boldsymbol{c}, \boldsymbol{d}, I(\boldsymbol{e}))) \approx_c (M, \boldsymbol{c}, \boldsymbol{d} = \mathrm{A}(M, \boldsymbol{c}), S(T, \boldsymbol{c}, \boldsymbol{d}, I')). \tag{11}$$

Indeed, if a distinguisher D distinguishes between the above distributions with some non-negligible advantage δ, then one can break Assumption A.1 as follows. Define the adversary $\mathrm{A}^*(M, \boldsymbol{u})$ that first computes $\boldsymbol{c} = \boldsymbol{u} + 0^u \circ \mathsf{Ecc}(\boldsymbol{a}_k)$ and then generates $\boldsymbol{d} = \mathrm{A}(x, \boldsymbol{b}, M, \boldsymbol{c})$. Next consider the mapping that takes $M, \boldsymbol{u}$ and a bit σ and outputs the tuple $(M, \boldsymbol{c} := \boldsymbol{u} + 0^u \circ \mathsf{Ecc}(\boldsymbol{a}_k), \boldsymbol{d}, \sigma)$. This mapping takes the distribution in the LHS of Eq. (7) to the distribution in LHS of Eq. (11), and takes the RHS of Eq. (7) to the distribution in RHS of Eq. (11), therefore D can be used to distinguish with advantage δ between the LHS and the RHS of Eq. (7), in contradiction to Assumption (A.1). □

Hybrid H_3. Our last Hybrid H_3 is defined similarly to H_2, except that instead of applying Step 4, we set the output of Bob to be $\boldsymbol{y}$ which is defined as in the real protocol to be the solution of the system

$$\boldsymbol{d}[I] = T[I] \begin{pmatrix} \boldsymbol{r}'' \\ \boldsymbol{y} \end{pmatrix}. \tag{12}$$

We show that the outcome $(M, \boldsymbol{c}, \boldsymbol{y})$, of H_3, is computationally indistinguishable from the outcome, $(M, \boldsymbol{c}, \boldsymbol{y}'')$, of H_2. Indeed, fix some $M, \boldsymbol{c}$ and $\boldsymbol{d} = \mathrm{A}(M, \boldsymbol{c})$, and condition on the event that the span condition holds, i.e., $\boldsymbol{d}[I] \in \mathsf{colspan}\left((T|\boldsymbol{c})[I]\right)$. (If this condition does not hold then the two hybrids are identically distributed.) It suffices to show that $\boldsymbol{y} = x'\boldsymbol{a}_k + \boldsymbol{b}'$ where $x', \boldsymbol{b}'$ are the solutions of the system

$$\boldsymbol{d}[I'] = T[I'] \begin{pmatrix} \boldsymbol{r}' \\ \boldsymbol{b}' \end{pmatrix} + x'\boldsymbol{c}[I']. \tag{13}$$

Let $I^* = I \cap I'$ and recall that $\boldsymbol{c}[I^*] = T[I^*] \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{a}_k \end{pmatrix}$ and so

$$\boldsymbol{d}[I^*] = T[I^*] \begin{pmatrix} \boldsymbol{r}' + x'\boldsymbol{r} \\ \boldsymbol{b}' + x'\boldsymbol{a}_k \end{pmatrix} = T[I^*] \begin{pmatrix} \boldsymbol{r}'' \\ \boldsymbol{y} \end{pmatrix}. \tag{14}$$

We complete the argument by noting that, except with negligible probability, $T[I^*]$ has full rank. To prove this it suffices to show that both $M_{\mathrm{top}}[I^*_{\mathrm{top}}]$ and $\mathsf{Ecc}[I^*_{\mathrm{bot}}]$ have full rank. Indeed, this follows by noting that $I^* \leftarrow \mathsf{BER}^m(1-\mu')$ for $\mu' = 2\mu - \mu^2$ and by using the modified linear independence property of the matrix M (property (a)) and by using a Chernoff bound on the erasures fraction and using the assumption on Ecc (property (b) as stated at the beginning of the section). □

B About the Correlated Noisy Codeword Assumption

We provide some evidence towards the validity of Assumption 6.1. Specifically, we show that, over the binary field $\mathbb{F}_2$, the failure of Assumption 6.1 allows us to partially reduce the noise in a Search-LPN instance, and is therefore unlikely to be true.

Formally, for a distribution $\mathcal{T} = \{\mathcal{T}_k\}$ over binary $m(k) \times n(n)$ matrices where $m(k), n(k)$ are some polynomials in the security parameter k and weight parameter $w(k) < m(k)$, let us denote by $\mathsf{SLPN}(\mathcal{T}, w)$ the problem whose input is a matrix $T \leftarrow \mathcal{T}_k$ and a random noisy vector $\boldsymbol{c} = T\boldsymbol{v} + \boldsymbol{e}$ where $\boldsymbol{v} \leftarrow \mathbb{F}_2^{n(k)}$, and $\boldsymbol{e} \in \mathbb{F}_2^{m(k)}$ is a random vector of Hamming weight $w(k)$, and the goal is to recover the noise vector $\boldsymbol{e}$. (Note that this is equivalent to recovering the information word $\boldsymbol{v}$.) We let $\mathsf{SLPN}'(\mathcal{T}, w)$ denote the variant of the problem where the noise $\boldsymbol{e}$ is an arbitrary vector whose Hamming weight is at most μ and $\boldsymbol{v}$ is an arbitrary vector in $\mathbb{F}_2^n$.[33] To make the problem well-defined, we implicitly assume that, except with negligible probability, a random matrix sampled from $\mathcal{T}_k$ has a distance of $2w(k)$. Thus unique decoding is possible.

Proposition B.1. *Suppose that there exists an efficient algorithm A, some function $\ell(k) = \omega(\log k)$, some inverse polynomial $\delta(k)$, and an infinite set K of integers, such that for every $k \in K$*

$$\Pr_{\boldsymbol{d} \leftarrow \mathrm{A}^*(\boldsymbol{c})} \left[\boldsymbol{d}[I] \in \mathsf{colspan}\left(T[I]\right) \quad \textit{and} \quad \rho\left(\boldsymbol{d}, \mathsf{colspan}\left(T|\boldsymbol{c}\right)\right) = \ell \right] \geq \delta(k)$$

where $T \leftarrow \mathcal{T}$ and $\boldsymbol{c} = T\boldsymbol{v} + \boldsymbol{e}$ for $\boldsymbol{v} \leftarrow \mathbb{F}_2^n$, and $\boldsymbol{e} \in \mathbb{F}_2^m$ is a random vector of Hamming weight w, and $I = I(\boldsymbol{e})$

Then, there exists an efficient reduction B that, with probability δ, maps $\mathsf{SLPN}(\mathcal{T}, \mu)$ to 2 instances of $\mathsf{SLPN}'(\mathcal{T}, w - \ell)$ for all $k \in K$.

Proof. Given $(T, \boldsymbol{c})$, the algorithm B invokes A and gets $\boldsymbol{d}$. Next, B calls an $\mathsf{SLPN}'(\mathcal{T}, w-\ell)$ solver on $(\boldsymbol{T}, \boldsymbol{d})$ and get as a result a noise vector $\boldsymbol{e}'$. The algorithm B proceeds with a query $(\boldsymbol{T}, \boldsymbol{c} - \boldsymbol{e}')$ to the $\mathsf{SLPN}'(\mathcal{T}, w - \ell)$-solver and receives a noise vector $\boldsymbol{e}''$. Finally, B outputs the noise vector $\boldsymbol{e}' + \boldsymbol{e}''$.

To analyze the reduction, let us write $\boldsymbol{c}$ as $T\boldsymbol{v} + \boldsymbol{e}$ where $\boldsymbol{e}$ is a random vector of weight w, let $I = I(\boldsymbol{e})$ be the set of zero (non-noisy) coordinates of $\boldsymbol{e}$. Assuming that A succeeds, we can write $\boldsymbol{d} = T\boldsymbol{v}' + \boldsymbol{e}'$ where $\boldsymbol{e}'[I] = 0^{|I|} = 0^{m-w}$. Let $\overline{I}$ denote the complement of I. Then, $\boldsymbol{e}'[\overline{I}]$ has (a) at least ℓ ones and (b) at least ℓ zeroes. Otherwise, $\boldsymbol{d}$ is ℓ-close to either $T\boldsymbol{v}'$ or to $T\boldsymbol{v}' + \boldsymbol{e}$, contradicting the fact that $\boldsymbol{d}$ is ℓ-far from $\mathsf{colspan}\left(T|\boldsymbol{c}\right)$. By (b), the distance between the vector $\boldsymbol{d}$ to the column-span of $\boldsymbol{T}$ is at most is $w - \ell$, and so $(\boldsymbol{T}, \boldsymbol{d})$ is a valid instance of $\mathsf{SLPN}'(\mathcal{T}, w - \ell)$. Now given the solution $\boldsymbol{e}'$ to the first query, it follows from (a) that the distance between the vector $\boldsymbol{c} - \boldsymbol{e}' = T\boldsymbol{v} + \boldsymbol{e} - \boldsymbol{e}'$ and the column-span of $\boldsymbol{T}$ is at most $(w - \ell)$. Hence, $(\boldsymbol{T}, \boldsymbol{c} - \boldsymbol{e}')$ is a valid instance of $\mathsf{SLPN}'(\mathcal{T}, w - \ell)$. Observe that $\boldsymbol{c} - \boldsymbol{e}'$ can be written as $T\boldsymbol{v} + (\boldsymbol{e} - \boldsymbol{e}')$ and that $(\boldsymbol{e} - \boldsymbol{e}')[I] = 0^{|m-w|}$. Therefore, the $\mathsf{SLPN}'(\mathcal{T}, w - \ell)$-solver must return $\boldsymbol{e}'' = (\boldsymbol{e} - \boldsymbol{e}')$, which completes the proof. □

Thus with polynomial success probability, we reduce the distance by $\omega(\log k)$. Such a reduction seems unlikely given the state-of-the-art algorithms for SearchLPN.

Remark 9. *For simplicity, we assumed that A violates a variant of Assumption 6.1 in which the noise vector is sampled uniformly over all vectors of weight w instead of using a sequence of Bernoulli random variables of mean $\mu = w/m$ as in Section 6. This difference is not crucial, and one can show that if A violates Assumption 6.1 wrt a noise vector that is sampled from a Bernoulli distribution with constant parameter μ, then A also violates the assumption over fixed-weight noise of some magnitude $w(k) \in (\mu m(k) \pm (m(k))^{2/3})$. To see this, observe that the Bernoulli noise distribution is*

[33] In fact, this variant is equivalent to the variant in which $\boldsymbol{v}$ is sampled uniformly, due to the random-self reduction of LPN; see [BFKL93].

a convex combination of fixed-weight noise distributions with different weight parameters. Moreover, by a Chernoff-bound, with all but negligible probability, the weight of the noise vector is close to the expectation $\mu m(k)$. Alternatively, one can employ the fixed-weight noise distribution throughout the paper.