Compressing Vector OLE

Elette Boyle
IDC, Israel
eboyle@alum.mit.edu

Geoffroy Couteau
KIT, Germany
geoffroy.couteau@kit.edu

Niv Gilboa
Ben Gurion University, Israel
gilboan@bgu.ac.il

Yuval Ishai
Technion, Israel
yuvali@cs.technion.ac.il

ABSTRACT

Oblivious linear-function evaluation (OLE) is a secure two-party protocol allowing a receiver to learn any linear combination of a pair of field elements held by a sender. OLE serves as a common building block for secure computation of arithmetic circuits, analogously to the role of oblivious transfer (OT) for boolean circuits.

A useful extension of OLE is *vector* OLE (VOLE), allowing the receiver to learn any linear combination of two *vectors* held by the sender. In several applications of OLE, one can replace a large number of instances of OLE by a smaller number of instances of VOLE. This motivates the goal of amortizing the cost of generating long instances of VOLE.

We suggest a new approach for fast generation of pseudo-random instances of VOLE via a deterministic local expansion of a pair of short correlated seeds and no interaction. This provides the first example of compressing a non-trivial and cryptographically useful correlation with good concrete efficiency. Our VOLE generators can be used to enhance the efficiency of a host of cryptographic applications. These include secure arithmetic computation and non-interactive zero-knowledge proofs with reusable preprocessing.

Our VOLE generators are based on a novel combination of function secret sharing (FSS) for multi-point functions and linear codes in which decoding is intractable. Their security can be based on variants of the learning parity with noise (LPN) assumption over large fields that resist known attacks. We provide several constructions that offer tradeoffs between different efficiency measures and the underlying intractability assumptions.

CCS CONCEPTS

• **Security and privacy → Cryptography**;

KEYWORDS

Secure computation, correlation generators, FSS, OLE, LPN, NIZK

CCS '18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243868

1 INTRODUCTION

Secret correlated randomness is a valuable resource for cryptographic protocols. For instance, a pair of identical secret random strings can be used for fast and perfectly secure communication, and more complex correlations such as "multiplication triples" [9, 12, 25] provide an analogous speedup for secure computation. A major difference between these two types of correlations is that while the former can be easily expanded locally from a short common seed by using any pseudorandom generator, it seems much harder to apply a similar compression procedure to the latter without compromising security.

More generally, consider the following loosely defined notion of a *pseudorandom correlation generator.* For a "long" target two-party correlation (Z_0, Z_1), we would like to locally expand a pair of correlated "short" strings $(\mathsf{seed}_0, \mathsf{seed}_1)$ into a pair of outputs (z_0, z_1), where $z_0 = \mathsf{Expand}(\mathsf{seed}_0)$ and $z_1 = \mathsf{Expand}(\mathsf{seed}_1)$. This should be done so that the joint output is indistinguishable from (Z_0, Z_1) not only to the outside world, but also to an *insider* who learns one seed seed_b and is trying to infer information about the other output z_{1-b} beyond what is implied by its output z_b.

For non-trivial two-party correlations, correlation generators as above were only constructed using indistinguishability obfuscation [44], homomorphic secret sharing [16], and key-homomorphic pseudorandom functions [65]. However, despite optimization efforts, none of these constructions is sufficiently efficient to offer a competitive alternative to traditional interactive protocols.

The focus of this work is on a special type of correlation related to *oblivious linear-function evaluation* (OLE). The OLE functionality allows a receiver to learn any linear combination of two field elements held by a sender. OLE is a common building block for secure computation of arithmetic circuits [28, 52, 62], analogously to the role of oblivious transfer (OT) for boolean circuits [28, 42, 51, 54].

A useful extension of OLE is *vector* OLE (VOLE), allowing the receiver to learn any linear combination of two *vectors* held by the sender. In several applications of OLE, one can replace a large number of instances of OLE by a small number of long instances of VOLE [4]. This motivates the goal of amortizing the cost of implementing long VOLE. Despite recent progress (see Section 1.3 below), the concrete communication and computation costs of the best VOLE protocols still leave much to be desired.

Motivated by the above goal, we study the question of compressing a random VOLE correlation, or *VOLE correlation* for short. In a VOLE correlation of length n over a finite field $\mathbb{F}$, the sender P_0 obtains a pair of random vectors $Z_0 = (\vec{u}, \vec{v})$, where $\vec{u}$ and $\vec{v}$ are uniformly distributed over $\mathbb{F}^n$, and the receiver P_1 obtains a random linear combination of the two vectors, namely $Z_1 = (x, \vec{u}x + \vec{v})$ for $x \in_R \mathbb{F}$. A VOLE correlation can be used to realize the VOLE functionality via a simple and efficient protocol, similarly to protocol implementing string OT from a random string OT [10]. In fact, string OT is *equivalent* to VOLE over the field $\mathbb{F} = \mathbb{F}_2$.

A natural approach for generating a VOLE correlation is via reduction to random string OT. Indeed, random string OT correlation can be easily compressed using any pseudorandom generator (PRG), and moreover a length-n VOLE over $\mathbb{F}$ can be realized with perfect security (against a semi-honest adversary) using $\ell = \lceil \log_2 \mathbb{F} \rceil$ instances of string OT of length $n\ell$ each [38]. The factor-ℓ communication overhead of this reduction can be significant for computations over large fields, which often arise in applications. But more importantly, the construction of VOLE from string OT requires the sender to feed the OT oracle with *correlated* random strings, even when the goal is to obtain a random instance of VOLE. This correlation makes the natural reduction of random VOLE to random string OT fail in the non-interactive setting we consider here.

1.1 Our Contribution

We give simple and efficient constructions of VOLE correlation generators based on conservative variants of the Learning Parity with Noise (LPN) assumption over large fields.[1] As far as we know, our work gives the first non-trivial example for a useful correlation generator with good concrete efficiency.

To give just one example, we estimate that for a field $\mathbb{F}$ with $\lceil \log_2 |\mathbb{F}| \rceil = 128$, we can generate a length-10^6 VOLE correlation from a pair of correlated seeds whose length is less than 1000 field elements using less than 25 milliseconds of local computation on a standard laptop using a single core and a common GPU.

Our VOLE generators can be useful in a variety of cryptographic applications. We discuss a few such applications below.

Rate 1/2 VOLE**.** As a direct application, we get a standard VOLE protocol in the plain model with unique efficiency features. This protocol is obtained by using general-purpose (OT-based) secure two-party computation to distribute the seed generation, locally expanding the seeds, and then using the simple reduction from VOLE to random VOLE. The protocol has asymptotic rate 1/2 (namely, the asymptotic communication complexity is dominated by communicating $2n$ field elements) and almost the entire computational work can be performed offline, following the seed generation, without any interaction. Beyond its direct efficiency benefits, this "local preprocessing" feature has several other advantages, including the ability to make decisions about who to interact with in the future (and how much) without revealing these decisions to the outside world. See [16] for further discussion. Our protocol can be compared to the recent VOLE protocol from [4], which under similar assumptions achieves rate 1/3 and does not enjoy the local preprocessing feature. An additional unique feature of our protocol (unlike other VOLE protocols from the literature) is that achieving security against malicious parties has vanishing amortized cost. As long as the seed generation sub-protocol is secure against malicious parties, the entire VOLE protocol is secure.

Secure arithmetic computation and beyond. Our efficient implementation of VOLE can serve as a useful building block in secure computation protocols. For instance, given an additively shared scalar $x \in \mathbb{F}$ and an additively shared vector $\vec{u} \in \mathbb{F}^n$, one can securely compute an additive sharing of $\vec{u}x$ via two invocations of length-n VOLE. Such scalar-vector multiplications are common in applications that involve linear algebra. See [4, 28, 52, 53, 61] and references therein. More generally, VOLE is useful for secure computation of arithmetic circuits in which multiplication gates have a large fan-out, as well as round-efficient secure arithmetic computation via arithmetic garbling [5]. Finally, VOLE can be helpful even for secure computation tasks that are not arithmetic in nature. For instance, OLE has been applied for efficiently realizing secure keyword search [34] and set intersection [37]. These applications can benefit from long instances of VOLE, e.g., when securely computing the intersection of one set with many other sets.

NIZK with reusable setup. Finally, we demonstrate the usefulness of VOLE generators in the context of non-interactive zero-knowledge proofs (NIZK). We consider the following setting for NIZK with reusable interactive setup. In an offline setup phase, before the statements to be proved are known, the prover and the verifier interact to securely generate correlated random seeds. The seeds can then be used to prove any polynomial number of statements by having the prover send a single message to the verifier for each statement. In this setting, we can leverage our fast VOLE generators towards NIZK proofs for arithmetic circuit satisfiability in which the proof computation and verification involve just a small number of field operations per gate, and the setup cost is comparable to the circuit complexity of (a single instance of) the verification predicate.

Our NIZK protocols are based on simple *honest-verifier* zero-knowledge protocols for arithmetic circuit satisfiability that consist of parallel calls to VOLE, where the honest verifier's VOLE inputs are independent of the statement being proved. Such protocols, in turn, can be obtained from linear PCPs for circuit satisfiability [14, 35, 47]. This application of VOLE generators crucially relies on the field being large for eliminating selective failure attacks. (Similar NIZK protocols based on OT [50, 55] are not fully reusable because they are susceptible to such attacks.) The honest-verifier VOLE-based NIZK protocols we use are simplified variants of a NIZK protocol from [21], which provides security against malicious verifiers using only parallel calls to VOLE and no additional interaction. The price we pay for the extra simplicity is that our setup phase needs to rely on general-purpose interactive MPC for ensuring that the verifier's (reusable) VOLE inputs are well formed.

We conclude by summarizing the two advantages of VOLE correlation over the string OT correlation which is easier to generate. A *quantitative* advantage is that VOLE natively supports arithmetic computations without the $\log_2 |\mathbb{F}|$ communication overhead of the OT-based approach discussed above. A *qualitative* advantage is that in certain applications (such as the NIZK protocol from [21] and our honest-verifier variants), VOLE can be used to eliminate selective failure attacks by ensuring that every adversarial strategy is either harmless or leads to failure with overwhelming probability.

[1]Roughly speaking, the LPN assumption says that in a random linear code, a noisy random codeword is pseudo-random. It can be equivalently formulated by requiring that the *syndrome* of a random low-weight noise vector is pseudo-random. Our constructions require a slightly sub-constant noise rate, but otherwise can be quite flexible about the choice of the code and its information rate. See Section 2.3 for more details.

1.2 Overview of the Techniques

Our VOLE generators are based on a novel combination of function secret sharing (FSS) [17] and noisy linear encodings. For the purpose of explaining the technique, it is convenient to view a VOLE correlation as a "shared vector-scalar product." That is, the sender knows a random vector $\vec{u} \in \mathbb{F}^n$, the receiver knows a random scalar $x \in \mathbb{F}$, and they both hold additive shares of $\vec{u}x$. The key idea is that efficient PRG-based FSS techniques allow compressing this correlation in the special case where $\vec{u}$ is sparse, namely it has few nonzero entries. However, this alone is not enough, since $\vec{u}$ must be pseudorandom to the receiver, which is certainly not the case for a sparse vector.

To convert "sparse" to "pseudorandom" we rely on the LPN assumption. This can be achieved in two different ways. In the *primal variant* of our construction, we achieve this by adding to the sparse $\vec{u}$ a random vector in a linear code C in which the LPN assumption is conjectured to hold. To do this, the sender gets a short message $\vec{a}$ and $\vec{a}x$ is shared between the parties. By locally applying the linear encoding of C to $\vec{a}$ and the shares of $\vec{a}x$, the VOLE correlation is maintained, except that the sparse $\vec{u}$ is masked with a random codeword $C(\vec{a})$ where both $\vec{u}$ and the codeword are unknown to the receiver. If C satisfies the LPN assumption with the level of noise corresponding to the sparsity of $\vec{u}$, the sum looks pseudorandom to the receiver.

The main advantage of the primal construction is that it is conjectured to be secure even with a code C that has constant locality, namely each codeword symbol is a linear combination of a constant number of message symbols [2, 4]. This enables fast incremental generation of VOLE, one entry at a time. Its main disadvantage is that its output size can be at most quadratic in the seed size. Indeed, a higher stretch would make it possible to guess a sufficiently large number of noiseless coordinates to allow efficient decoding via Guassian elimination.

To achieve an arbitrary polynomial stretch, one can use the *dual variant* of our construction. Here the parties shrink both the sparse $\vec{u}$ and the shares of $\vec{u}x$ by applying a public *compressive* linear mapping H. If H is a parity check matrix of a code for which LPN holds, the output of H looks pseudorandom even when given H. A disadvantage of the dual approach is that the compressive mapping H cannot have constant locality.

We propose several different optimizations of the above approaches. These include LPN-friendly mappings C and H that can be computed in linear time, improved implementations of the FSS component of the construction, and secure protocols for distributing the setup algorithm that generates the seeds. Under plausible variants of the LPN assumption, the asymptotic time complexity of the seed expansion is linear in the output size. We discuss further optimizations and give some concrete efficiency estimates in Section 5.

1.3 Related Work

The idea of compressing cryptographically useful correlations was first put forward in [39], who focused on the case of multi-party correlations that are distributed uniformly over a linear space. This idea was generalized in [23]. The problem of compressing useful two-party correlations was studied in [16], who presented solutions that rely on "group-based" homomorphic secret sharing. However, the compression schemes proposed from [16] have poor concrete efficiency, despite significant optimization efforts.

Variants of the LPN assumption were used as a basis for secure arithmetic computation in several previous works [4, 28, 52, 62]. The core idea is to use the homomorphic property of a linear code to compute a linear function on a noisy encoded message, and then filter out the noisy coordinates using OT. This technique is quite different from ours. In particular, it inherently relies on erasure-decoding which we completely avoid. This gives us more freedom in choosing efficiently encodable (or checkable) codes in which LPN resists known attacks.

2 PRELIMINARIES

We consider algorithms that take inputs and produce outputs from a finite field $\mathbb{F}$ or finite Abelian group $\mathbb{G}$. All of our protocols are fully *arithmetic* in that they only require a black-box access to the underlying algebraic structure in the same sense as in [4, 52]. In particular, the number of arithmetic operations performed by our protocols does not grow with the field or group size. By default vectors $\vec{v}$ are interpreted as row vectors.

2.1 Vector OLE

Vector OLE (VOLE) is the arithmetic analogue of string OT. Concretely, the VOLE functionality is a two-party functionality that takes a pair of vectors from the *sender* P_0, and allows the *receiver* P_1 to learn a chosen linear combination of these vectors. More formally, given a finite field $\mathbb{F}$, the VOLE functionality takes a pair of vectors $(\vec{u}, \vec{v}) \in \mathbb{F}^n \times \mathbb{F}^n$ from P_0 and a scalar $x \in \mathbb{F}$ from P_1. It outputs $\vec{w} = \vec{u}x + \vec{v}$ to P_1. We will also consider a randomized version of VOLE where the sender's inputs $(\vec{u}, \vec{v})$ are picked at random by the functionality and delivered as outputs to the sender. The deterministic VOLE functionality can be easily reduced to the randomized one analogously to the reduction of OT to random OT [10] (see Section 6.1.1).

We note that our results can apply to generating VOLE over non-field rings (e.g., $\mathbb{Z}_{2^k}$) under suitable variants of the underlying intractability assumptions [52]. This can be useful in turn for secure arithmetic computation over rings [22, 24, 52]. For simplicity, we focus here on the case of VOLE over fields.

2.2 Function Secret Sharing

Informally, a function secret sharing (FSS) scheme [17] splits a function $f : I \to \mathbb{G}$ into two functions f_0 and f_1 such that $f_0(x) + f_1(x) = f(x)$ for every input x, and each f_b computationally hides f. In this work we rely on efficient constructions of FSS schemes for simple classes of functions, including multi-point functions and comparison functions.

Definition 2.1 (Adapted from [18]). A 2-party *function secret sharing* (FSS) scheme for a class of functions $\mathcal{F} = \{f : I \to \mathbb{G}\}$ with input domain I and output domain an abelian group $(\mathbb{G}, +)$, is a pair of PPT algorithms FSS = (FSS.Gen, FSS.Eval) with the following syntax:

- FSS.Gen$(1^\lambda, f)$, given security parameter λ and description of a function $f \in \mathcal{F}$, outputs a pair of keys (K_0, K_1);

- $\mathsf{FSS.Eval}(b, K_b, x)$, given party index $b \in \{0,1\}$, key K_b, and input $x \in I$, outputs a group element $y_b \in \mathbb{G}$.

Given an allowable leakage function $\mathsf{Leak} : \{0,1\}^* \rightarrow \{0,1\}^*$, the scheme FSS should satisfy the following requirements:

- **Correctness.** For any $f : I \rightarrow \mathbb{G}$ in $\mathcal{F}$ and $x \in I$, we have $\Pr[(K_0, K_1) \xleftarrow{R} \mathsf{FSS.Gen}(1^\lambda, f) : \sum_{b\in\{0,1\}} \mathsf{FSS.Eval}(b, K_b, x) = f(x)] = 1$.
- **Security.** For any $b \in \{0,1\}$, there exists a PPT simulator Sim such that for any polynomial-size function sequence $f_\lambda \in \mathcal{F}$, the distributions $\{(K_0, K_1) \xleftarrow{R} \mathsf{FSS.Gen}(1^\lambda, f_\lambda) : K_b\}$ and $\{K_b \xleftarrow{R} \mathsf{Sim}(1^\lambda, \mathsf{Leak}(f_\lambda))\}$ are computationally indistinguishable.

Unless otherwise specified, we assume that for $f : I \rightarrow \mathbb{G}$, the allowable leakage $\mathsf{Leak}(f)$ outputs $(I, \mathbb{G})$, namely a description of the input and output domains of f.

Some applications of FSS require applying the evaluation algorithm on *all inputs*. Given an FSS (FSS.Gen, FSS.Eval), we denote by FSS.FullEval an algorithm which, on input a bit b, and an evaluation key K_b, outputs a list of $|I|$ elements of $\mathbb{G}$ corresponding to the evaluation of $\mathsf{FSS.Eval}(b, K_b, \cdot)$ on every input $x \in I$ (in some arbitrary specified order). While FSS.FullEval can always be realized with $|I|$ invocations of FSS.Eval, it is typically possible to obtain a more efficient construction. Below, we recall some results from [18] on FSS schemes for useful classes of functions.

2.2.1 Distributed Point Functions. A distributed point function (DPF) [40] is an FSS scheme for the class of point functions $f_{\alpha,\beta} : \{0,1\}^\ell \rightarrow \mathbb{G}$ which satisfy $f_{\alpha,\beta}(\alpha) = \beta$, and $f_{\alpha,\beta}(x) = 0$ for any $x \neq \alpha$. A sequence of works [17, 18, 40] has led to highly efficient constructions of DPF schemes from any pseudorandom generator (PRG), which can be implemented in practice using block ciphers such as AES.

Theorem 2.2 ([18]). *Given a PRG $G : \{0,1\}^\lambda \rightarrow \{0,1\}^{2\lambda+2}$, there exists a DPF for point functions $f_{\alpha,\beta} : \{0,1\}^\ell \rightarrow \mathbb{G}$ with key size $\ell \cdot (\lambda + 2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil$ bits. For $m = \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil$, the key generation algorithm* Gen *invokes G at most $2(\ell + m)$ times, the evaluation algorithm* Eval *invokes G at most $\ell + m$ times, and the full evaluation algorithm* FullEval *invokes G at most $2^\ell(1 + m)$ times.*

Note that a naive construction of FullEval from Eval would require $2^\ell(\ell + m)$ invocations of G.

2.2.2 FSS for Multi-Point Functions. Our results crucially rely on FSS schemes for multi-point functions, a natural generalization of point functions. A t-point function evaluates to 0 everywhere, except on t specified points. When specifying multi-point functions we often view the domain of the function as $[n]$ for $n = 2^\ell$ instead of $\{0,1\}^\ell$. Formally:

Definition 2.3 (Multi-Point Function). An (n, t)-multi-point function over an abelian group $(\mathbb{G}, +)$ is a function $f_{S,\vec{y}} : [n] \rightarrow \mathbb{G}$, where $S = \{s_1, \cdots, s_t\}$ is a subset of $[n]$ of size t, $\vec{y} = (y_1, \cdots, y_t) \in \mathbb{G}^{kt}$, and $f_{S,\vec{y}}(s_i) = y_i$ for any $i \in [t]$, and $f_{S,y}(x) = 0$ for any $x \in [n] \setminus S$.

We assume that the description of S includes the input domain $[n]$ so that $f_{S,\vec{y}}$ is fully specified.

A *Multi-Point Function Secret Sharing* (MPFSS) is an FSS scheme for the class of multi-point functions, where a point function $f_{S,\vec{y}}$ is represented in a natural way. An MPFSS can be easily obtained by adding t instances of DPF. We discuss optimizations of this simple MPFSS construction in Section 4.

We assume that an MPFSS scheme leaks not only the input and output domains but also the number of points t that the multi-point function specifies.

2.3 Learning Parity with Noise

Our constructions rely on variants of the Learning Parity with Noise (LPN) assumption over large fields. Similar assumptions have been previously used in the context of secure arithmetic computation [4, 28, 36, 52, 62]. Unlike most of these works, the flavors of LPN on which we rely do not require the underlying code to have an algebraic structure and are thus not susceptible to algebraic (list-) decoding attacks.

For a finite field $\mathbb{F}$, we denote by $\mathsf{Ber}_r(\mathbb{F})$ the Bernoulli distribution obtained by sampling a uniformly random element of $\mathbb{F}$ with probability r, and 0 with probability $1 - r$. We define below the Learning Parity with Noise assumption over a field $\mathbb{F}$.

Definition 2.4. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(k, q, \mathbb{F})$ outputs (a description of) a matrix $A \in \mathbb{F}^{k\times q}$. For dimension $k = k(\lambda)$, number of queries (or block length) $q = q(\lambda)$, and noise rate $r = r(\lambda)$, the $\mathsf{LPN}(k, q, r)$ assumption with respect to $\mathbf{C}$ states that for any polynomial-time non-uniform adversary $\mathcal{A}$, it holds that

$$\Pr[\mathbb{F} \leftarrow \mathcal{A}(1^\lambda), A \xleftarrow{R} \mathbf{C}(k, q, \mathbb{F}), \vec{e} \xleftarrow{R} \mathsf{Ber}_r(\mathbb{F})^q,$$
$$\vec{s} \xleftarrow{R} \mathbb{F}^k, \vec{b} \leftarrow \vec{s} \cdot A + \vec{e} : \mathcal{A}(A, \vec{b}) = 1]$$
$$\approx \Pr[\mathbb{F} \leftarrow \mathcal{A}(1^\lambda), A \xleftarrow{R} \mathbf{C}(k, q, \mathbb{F}), \vec{b} \xleftarrow{R} \mathbb{F}^q : \mathcal{A}(A, \vec{b}) = 1].$$

By default, we assume that $\mathbf{C}$ outputs a uniformly random matrix, but other distributions of codes will be used for better efficiency.

Note that the decision LPN assumption, given above, can be reduced in polynomial time to its search variant (where the attacker must find the secret vector $\vec{s}$). While this reduction is not tight, in practice, no better attack is known on decision LPN than on search LPN. Note also that the LPN assumption is equivalent to its dual version, which states that it is infeasible to distinguish $\vec{e} \cdot B$ from a random vector, where $\vec{e}$ is a noise vector and B is the *parity-check matrix* of the matrix $A \in \mathbb{F}^{k\times q}$ (i.e., B is a full-rank matrix in $\mathbb{F}^{q\times(q-k)}$ such that $A \cdot B = 0$). The equivalence to LPN follows immediately from the relation $\vec{e} \cdot B = (\vec{s} \cdot A + \vec{e}) \cdot B$ for any $\vec{s} \in \mathbb{F}^k$. The dual variant of LPN is also known as the *syndrome decoding problem.*

2.3.1 Attacks on the LPN Problem. In spite of its extensive use in cryptography, few cryptanalytic results are known for the general LPN assumption. We briefly outline below the main results; we refer the reader to [31] for a more comprehensive overview.

- **Gaussian elimination.** The most natural attack on LPN recovers $\vec{s}$ from $\vec{b} = \vec{s} \cdot A + \vec{e}$ by guessing k non-noisy coordinates of $\vec{b}$, and inverting the corresponding subsystem to verify whether the guess was correct. This approach recovers $\vec{s}$ in time at least $(1/(1-r))^k$ using at least $O(k/r)$

samples. For low-noise LPN, with noise rate $1/k^c$ for some constant $c \geq 1/2$, this translates to a bound on attacks of $O(e^{k^{1-c}})$ time using $O(k^{1+c})$ samples.

- **Information Set Decoding (ISD) [63].** Breaking LPN is equivalent to solving its dual variant, which can be interpreted as the task of decoding a random linear code. The best algorithms for this task are improvements of Prange's ISD algorithm, which attempts to find a size-w subset of the rows of B (the parity-check matrix of the code) that sums to $\vec{e} \cdot B$, where $w = rq$ is the number of noisy coordinates.
- **The BKW algorithm [15].** This algorithm is a variant of Gaussian elimination which achieves subexponential complexity even for high-noise LPN (e.g. constant noise rate), but requires a subexponential number of samples: the attack solves LPN over $\mathbb{F}_2$ in time $2^{O(k/\log(k/r))}$ using $2^{O(k/\log(k/r))}$ samples.
- **Combinations of the above [31].** The authors of [31] conducted an extended study of the security of LPN, and described combinations and refinements of the previous three attacks (called the *well-pooled Gauss attack*, the *hybrid attack*, and the *well-pooled MMT attack*). All these attacks achieve subexponential time complexity, but require as many sample as their time complexity.
- **Scaled-down BKW [58].** This algorithm is a variant of the BKW algorithm, tailored to LPN with polynomially-many samples. It solves LPN in time $2^{O(k/\log\log(k/r))}$, using $k^{1+\varepsilon}$ samples (for any constant $\varepsilon > 0$) and has worse performance in time and number of samples for larger fields.

In this paper, we will rely on the LPN assumption with high dimension k, low-noise (noise rate $1/k^\varepsilon$ for some constant ε), and a polynomially bounded number of samples ($q < k^2$, or even $q = k + o(k)$). We note that in this regime of parameters, no improvement is known over the standard Gaussian elimination attack, both in the asymptotic setting (BKW and the attacks of [31] require a subexponential number of samples, and the attack of [58] does not perform well on low-noise LPN), and in the concrete setting for any reasonable parameters (according to the detailed recent estimations of [31]). For a very limited number of samples (which is the case in our setting), variants of ISD are expected to provide relatively good results. However, they do not perform well in our specific scenario: when the LPN instance has high dimension and very low error rate ($r(\lambda) \to 0$ when $\lambda \to \infty$), according to the analysis of [69], all known variants of ISD (e.g. [11, 13, 33, 59, 60, 63, 67]) have essentially the same asymptotic complexity $2^{cw(1+o(1))}$ for a constant $c \approx -\log(1 - k/q)$ (with $w = rq$ the number of noisy coordinates). Therefore, their gain compared to the initial algorithm of Prange collapse in our setting. In all the concrete instances we consider, we estimated the security of the corresponding LPN instance using both Gaussian attacks and ISD (using the detailed concrete efficiency analysis of ISD given in [45]). In all situations, we found Gaussian elimination to perform better than ISD.

LPN-friendly codes. For the purpose of optimizing the computational complexity of LPN-based constructions, one can use a code generator **C** that outputs (the description of) a structured matrix A such that encoding is fast and yet LPN is still conjectured to hold. For instance, if A is a random Toeplitz matrix, encoding can be done in quasi-linear time but no better attacks on LPN are known compared to a random choice of A. There are in fact candidates for asymptotically good LPN-friendly codes that can be encoded by linear-size circuits over $\mathbb{F}$ [4, 29]. Finally, since we do not require our codes to have good minimal distance and our constructions do not require erasure-decoding, one can apply a heuristic randomized construction of LPN-friendly codes by composing a linear number of "simple" elementary linear operations (e.g., add to coordinate i a random multiple of coordinate j).

3 PSEUDORANDOM VOLE GENERATOR

In this section, we formally define our main notion of a pseudorandom VOLE generator (or VOLE generator for short), and provide two constructions that are dual to each other (in a sense that will be made formal). These constructions form the core technical contribution of our paper.

3.1 Defining VOLE Generator

Informally, a VOLE generator allows stretching a pair of short, correlated seeds into a long (pseudo)random VOLE, by locally applying a deterministic function Expand to the seeds. Defining the security notion for this primitive requires some care. Ideally, we would have liked to require that the protocol in which a trusted dealer distributes the seeds and the parties output the result of applying Expand to be a secure realization of the VOLE correlation according to the standard real vs. ideal paradigm for defining secure computation. However, as pointed out in [39], this security notion cannot be achieved in general. Intuitively, this stems from the fact that each party holds a *short representation* of its correlated string. For instance, consider a very simple correlation, where both parties should obtain the same long pseudorandom string. Then any generator for this correlation will reveal to the first party a short representation of the string of the other party, which cannot happen in an ideal implementation.

To overcome this issue, we rely on an alternative security notion, which roughly asserts the following. Consider the real-world experiment of distributing the two seeds and locally expanding them. We require that the seed seed_σ observed by party σ together with the expanded second output $\mathsf{Expand}(\mathsf{seed}_{1-\sigma})$ are indistinguishable from seed_σ together with a random output of party $1 - \sigma$ conditioned on $\mathsf{Expand}(\mathsf{seed}_\sigma)$ in a perfect VOLE correlation. We prove that this notion suffices for securely instantiating the standard protocol for computing a chosen-input VOLE from a random VOLE (see Section 6.1.1), and is hence sufficient for the applications we consider.

We allow the setup algorithm of the VOLE generator to fix the receiver's input x rather than choose it at random. This stronger flavor of VOLE generator, which is needed by some of the applications, is formalized below.

Definition 3.1 (Pseudorandom VOLE generator). A pseudorandom VOLE generator is a pair of algorithms (Setup, Expand) with the following syntax:

- $\mathsf{Setup}(1^\lambda, \mathbb{F}, n, x)$ is a PPT algorithm that given a security parameter λ, field $\mathbb{F}$, output length n, and scalar $x \in \mathbb{F}$ outputs a pair of seeds $(\mathsf{seed}_0, \mathsf{seed}_1)$, where seed_1 includes x;

- $\mathsf{Expand}(\sigma, \mathsf{seed}_\sigma)$ is a polynomial-time algorithm that given party index $\sigma \in \{0,1\}$ and a seed seed_σ, outputs a pair $(\vec{u}, \vec{v}) \in \mathbb{F}^n \times \mathbb{F}^n$ if $\sigma = 0$, or a vector $\vec{w} \in \mathbb{F}^n$ if $\sigma = 1$;

The algorithms (Setup, Expand) should satisfy the following:

- **Correctness.** For any field $\mathbb{F}$ and $x \in \mathbb{F}$, for any pair $(\mathsf{seed}_0, \mathsf{seed}_1)$ in the image of $\mathsf{Setup}(1^\lambda, \mathbb{F}, n, x)$ (for some n), denoting $(\vec{u}, \vec{v}) \leftarrow \mathsf{Expand}(0, \mathsf{seed}_0)$, and $\vec{w} \leftarrow \mathsf{Expand}(1, \mathsf{seed}_1)$, it holds that $\vec{u}x + \vec{v} = \vec{w}$.
- **Security.** For any (stateful, nonuniform) polynomial-time adversary $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{l}(\mathbb{F}, 1^n, x, x') \leftarrow \mathcal{A}(1^\lambda), \\ (\mathsf{seed}_0, \mathsf{seed}_1) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, x)\end{array} : \mathcal{A}(\mathsf{seed}_0) = 1\right]$$
$$\approx \Pr\left[\begin{array}{l}(\mathbb{F}, 1^n, x, x') \leftarrow \mathcal{A}(1^\lambda), \\ (\mathsf{seed}_0, \mathsf{seed}_1) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, x')\end{array} : \mathcal{A}(\mathsf{seed}_0) = 1\right].$$

Similarly, for any (stateful, nonuniform) adversary $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{l}(\mathbb{F}, 1^n, x) \leftarrow \mathcal{A}(1^\lambda), \\ (\mathsf{seed}_0, \mathsf{seed}_1) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, x), \\ (\vec{u}, \vec{v}) \leftarrow \mathsf{Expand}(0, \mathsf{seed}_0)\end{array} : \mathcal{A}(\vec{u}, \vec{v}, \mathsf{seed}_1) = 1\right]$$
$$\approx \Pr\left[\begin{array}{l}(\mathbb{F}, 1^n, x) \leftarrow \mathcal{A}(1^\lambda), \vec{u} \xleftarrow{R} \mathbb{F}^n, \\ (\mathsf{seed}_0, \mathsf{seed}_1) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, x), \\ \vec{w} \leftarrow \mathsf{Expand}(1, \mathsf{seed}_1), \vec{v} \leftarrow \vec{w} - \vec{u}x\end{array} : \mathcal{A}(\vec{u}, \vec{v}, \mathsf{seed}_1) = 1\right].$$

The reader might observe that one can trivially realize the above definition, simply by letting Setup directly output $\mathsf{seed}_0 \leftarrow (\vec{u}, \vec{v})$, and $\mathsf{seed}_1 \leftarrow \vec{u}x + \vec{v}$, and defining Expand to be the identity function. We will be interested in non-trivial realizations of VOLE generators, where the seed produced by Setup is *significantly shorter* than the number n of the pseudo-random VOLE instances being produced.

3.2 Primal VOLE Generator

We present the first of two VOLE generator constructions. To simplify the presentation, we introduce a "spreading function" spread_n (for any integer n) which takes as input a subset $S = \{s_1, \cdots, s_{|S|}\}$ of $[n]$ (with $s_1 < s_2 < \cdots < s_{|S|}$) and a vector $\vec{y} = (y_1, \cdots, y_{|S|}) \in \mathbb{F}^{|S|}$, such that $\mathsf{spread}_n(S, \vec{y})$ is the vector $\vec{z}$ satisfying $z_j = 0$ for any $j \in [n] \setminus S$, and $z_{s_i} = y_i$ for $i = 1$ to $|S|$. Note that the function $\mathsf{spread}_n(S, \cdot)$ is a linear function. Our construction of a pseudorandom VOLE generator G_{primal} is given in Figure 2.

Theorem 3.2. *Let $n = n(\lambda), k = k(\lambda), t = t(\lambda), \mathbb{F} = \mathbb{F}(\lambda)$ be such that $\mathsf{LPN}(k, n, t/n)$ holds over $\mathbb{F}$ with respect to the code with matrix $C_{k,n}$, and suppose MPFSS is a secure MPFSS scheme. Then G_{primal} is a secure VOLE generator.*

In the following, we prove Theorem 3.2.

3.2.1 Correctness.

By the MPFSS correctness, it holds that

$$\mathsf{MPFSS.FullEval}(0, K_0) + \mathsf{MPFSS.FullEval}(1, K_1) = \mathsf{spread}_n(S, x\vec{y}) = \vec{\mu}x.$$

VOLE **Generator** G_{primal}

- **Parameters:** dimension $k = k(\lambda)$, noise parameter $t = t(\lambda)$
- **Building blocks:** a code generator $\mathbf{C}$, such that $\mathbf{C}(k, n, \mathbb{F})$ defines a public matrix $C_{k,n} \in \mathbb{F}^{k \times n}$, and a multi-point function secret sharing MPFSS = (MPFSS.Gen, MPFSS.Eval, MPFSS.FullEval).
- $G_{\mathsf{primal}}.\mathsf{Setup}(1^\lambda, \mathbb{F}, n, x)$: pick a random size-t subset S of $[n]$, two random vectors $(\vec{a}, \vec{b}) \xleftarrow{R} \mathbb{F}^k \times \mathbb{F}^k$, and a random vector $\vec{y} \xleftarrow{R} \mathbb{F}^t$. Let $s_1 < s_2 < \cdots < s_t$ denote the elements of S. Set $\vec{c} \leftarrow \vec{a}x + \vec{b}$. Compute $(K_0, K_1) \xleftarrow{R} \mathsf{MPFSS.Gen}(1^\lambda, f_{S, x\vec{y}})$. Set $\mathsf{seed}_0 \leftarrow (\mathbb{F}, n, K_0, S, \vec{y}, \vec{a}, \vec{b})$ and $\mathsf{seed}_1 \leftarrow (\mathbb{F}, n, K_1, x, \vec{c})$. Output $(\mathsf{seed}_0, \mathsf{seed}_1)$.
- $G_{\mathsf{primal}}.\mathsf{Expand}(\sigma, \mathsf{seed}_\sigma)$:
If $\sigma = 0$, parse seed_0 as $(\mathbb{F}, n, K_0, S, \vec{y}, \vec{a}, \vec{b})$. Set $\vec{\mu} \leftarrow \mathsf{spread}_n(S, \vec{y})$. Compute $\vec{v_0} \leftarrow \mathsf{MPFSS.FullEval}(0, K_0)$. Output $(\vec{u}, \vec{v}) \leftarrow (\vec{a} \cdot C_{k,n} + \vec{\mu}, \vec{b} \cdot C_{k,n} - \vec{v_0})$.
If $\sigma = 1$, parse seed_1 as $(\mathbb{F}, n, K_1, x, \vec{c})$. Compute $\vec{v_1} \leftarrow \mathsf{MPFSS.FullEval}(1, K_1)$, and set $\vec{w} \leftarrow \vec{c} \cdot C_{k,n} + \vec{v_1}$. Output $\vec{w}$.

Figure 1: VOLE **Generator** G_{primal}

Therefore,

$$\begin{aligned}\vec{u}x + \vec{v} &= (\vec{a} \cdot C_{k,n} + \vec{\mu})x + \vec{b} \cdot C_{k,n} - \vec{v_0} \\ &= (\vec{a}x + \vec{b}) \cdot C_{k,n} + \vec{\mu}x - \mathsf{MPFSS.FullEval}(0, K_0) \\ &= \vec{c} \cdot C_{k,n} + \vec{\mu}x + \mathsf{MPFSS.FullEval}(1, K_1) - \vec{\mu}x \\ &= \vec{c} \cdot C_{k,n} + \vec{v_1} = \vec{w},\end{aligned}$$

which concludes the proof of correctness.

3.2.2 Security.

We start by proving that G_{primal} satisfies the first security requirement of VOLE generators under the secrecy property of the MPFSS. Recall that this first requirement states that no PPT adversary can distinguish the pair (seed_0, x) from (seed_0, x'), where $(\mathbb{F}, 1^n, x, x') \xleftarrow{R} \mathcal{A}(1^\lambda)$ and $(\mathsf{seed}_0, \mathsf{seed}_1) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, x)$, for a field $\mathbb{F}$ and a size parameter n chosen by $\mathcal{A}$. Note that the only part of $\mathsf{seed}_0 = (\mathbb{F}, n, K_0, S, \vec{y}, \vec{a}, \vec{b})$ which depends on x is the MPFSS key K_0. By the secrecy property of the MPFSS, there exists a simulator which, given only the allowable leakage $(\mathbb{F}, n, t)$, outputs a key K'_0 which is indistinguishable from K_0. As this simulator does not know any information about x, this immediately implies the first requirement.

We now turn our attention to the second requirement, which states that no efficient adversary $\mathcal{A}$ can distinguish $(\vec{u}, \vec{v}, \mathsf{seed}_1)$ from $(\vec{u'}, \vec{v'}, \mathsf{seed}_1)$, where $(\mathsf{seed}_0, \mathsf{seed}_1) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, x), (\vec{u}, \vec{v}) \leftarrow \mathsf{Expand}(0, \mathsf{seed}_0)$, $\vec{u'} \xleftarrow{R} \mathbb{F}^n$, and $\vec{v'} \leftarrow \mathsf{Expand}(1, \mathsf{seed}_1) - \vec{u'}x$, with $(\mathbb{F}, n, x)$ chosen by $\mathcal{A}$.

Let $\mathcal{A}$ be a stateful PPT adversary, and let $(\mathbb{F}, 1^n, x) \leftarrow \mathcal{A}(1^\lambda)$. We prove the second security requirement through a sequence of games.

- **Game 0.** Compute $(\mathsf{seed}_0, \mathsf{seed}_1) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, x)$, set $(\vec{u}, \vec{v}) \leftarrow \mathsf{Expand}(0, \mathsf{seed}_0)$, and send $(\vec{u}, \vec{v}, \mathsf{seed}_1)$ to $\mathcal{A}$. Denote β_0 the output of $\mathcal{A}$ in this game. Note that the input of $\mathcal{A}$ in this game is $\mathsf{seed}_1 = (\mathbb{F}, n, K_1, x, \vec{c})$, $\vec{u} = \vec{a} \cdot C_{k,n} + \vec{\mu}$, and $\vec{v} = \vec{b} \cdot C_{k,n} + \vec{v}_0 = \vec{b} \cdot C_{k,n} + \vec{v}_1 - \vec{\mu}x = \vec{c} \cdot C_{k,n} + \vec{v}_1 - (\vec{a} \cdot C_{k,n} + \vec{\mu})x$ (using the fact that $\vec{c} = \vec{a}x + \vec{b}$ and $\vec{v}_0 + \vec{v}_1 = \vec{\mu}x$).
- **Game 1.** In this game, compute the input of $\mathcal{A}$ as before, except that K_1 is now computed solely from $(\mathbb{F}, n, t)$ using the simulator for the secrecy of the MPFSS. Note that in this game, K_1 carries no information whatsoever about $\vec{\mu}$. Denote β_1 the output of $\mathcal{A}$ in this game; by the secrecy of the MPFSS, $|\Pr[\beta_1 = 1] - \Pr[\beta_0 = 1]| = \mathsf{negl}(\lambda)$.
- **Game 2.** In this game, pick $\vec{u'} \xleftarrow{R} \mathbb{F}^n$ and set $\vec{v'} \leftarrow \vec{c} \cdot C_{k,n} + \vec{v}_1 - \vec{u'}x = \mathsf{Expand}(1, \mathsf{seed}_1) - \vec{u'}x$. Note that the only difference between this game and the previous one is that we replaced $\vec{u} = \vec{a} \cdot C_{k,n} + \vec{\mu}$ by a uniformly random vector $\vec{u'}$. Observe that $\vec{u}$ is exactly a noisy linear encoding of $\vec{a}$, using the linear code $C_{k,n} \in \mathbb{F}^{k(\lambda) \times n}$, with noise vector $\vec{\mu}$. Since seed_1 carries no information about $\vec{\mu}$, $\vec{u}$ is therefore a noisy linear encoding of $\vec{a}$, where the number of noisy coordinates is exactly $t(\lambda)$ (as $\vec{\mu} = \mathsf{spread}_n(S, \vec{y})$ and $|\vec{y}| = k$), and each noisy coordinate is masked by a uniformly random element of $\mathbb{F}$. Therefore, distinguishing Game 2 from Game 1 is equivalent to breaking the LPN assumption of dimension $k(\lambda)$ over $\mathbb{F}$, with n samples and a noise rate $t(\lambda)/n$: denoting β_2 the output of $\mathcal{A}$ in this game, under the $\mathsf{LPN}(k(\lambda), n, t(\lambda)/n)$ assumption over $\mathbb{F}$, $|\Pr[\beta_1 = 1] - \Pr[\beta_2 = 1]| = \mathsf{negl}(\lambda)$; this concludes the proof of security of G_{primal}.

3.2.3 Efficiency. Instantiating the MPFSS with the PRG-based construction outlined in Section 2.2.2, the setup algorithm of G_{primal} outputs seeds of size $t \cdot (\lceil \log n \rceil (\lambda + 2) + \lambda) + (t + k) \cdot \log_2 |\mathbb{F}| = \tilde{O}(\lambda \cdot (k + t))$ for a field of size $|\mathbb{F}| = 2^{O(\lambda)}$. The best known attack on $\mathsf{LPN}(k, n, t/n)$ is the Gaussian elimination attack, which takes time $O((1 - t/n)^k)$. This implies that, over a large field $\mathbb{F}$ (such that $\log_2 |\mathbb{F}| \geq \lambda$), the optimal expansion factor is obtained by setting $k = t = O(n^{1/2+\varepsilon})$ for some $\varepsilon > 0$, in which case the Expand algorithm of the VOLE generator expands a seed of size $\tilde{O}(n^{1/2+\varepsilon})$ into a pseudorandom VOLE of size $O(n)$ (counting size as a number of elements of $\mathbb{F}$), and the best known attack takes subexponential time $O(e^{n^{2\varepsilon}})$. Regarding computational efficiency, expanding the seed requires $O((k + t) \cdot n)$ arithmetic operations, and $t \cdot n$ PRG evaluations.

Instantiating G_{primal} with parameters (k, n, t) over a field $\mathbb{F}$ yields a VOLE generator with seed length $t \cdot (\lceil \log n \rceil (\lambda + 2) + \lambda) + (t + k) \cdot \log_2 |\mathbb{F}|$ bits and output length $2n$ group elements (for $\mathsf{Expand}(0, \cdot)$) or n group elements (for $\mathsf{Expand}(1, \cdot)$). This VOLE generator is (T, ε)-secure iff $\mathsf{LPN}(k, n, t/n)$ with code $C_{k,n}$ is (T', ε)-secure and the MPFSS is (T'', ε)-secure, with $T' = T - O((k+t) \cdot n \cdot \log_2 |\mathbb{F}| + t \cdot n \cdot \lambda)$ and $T'' = T - O((k + t) \cdot n \cdot \log_2 |\mathbb{F}|)$.

A downside of this approach is that the expansion factor of the VOLE generator is limited to subquadratic. Below, we describe an alternative "dual" approach which overcomes this limitation and allows for an arbitrary polynomial expansion.

VOLE **Generator** G_{dual}

- **Parameters:** noise parameter $t = t(\lambda)$.
- **Building blocks:** a (dual) code generator $\mathbf{C}'$ (which generates on input $(n, n', \mathbb{F})$ a public matrix $H_{n',n} \in \mathbb{F}^{n' \times n}$, a random matrix by default), and a multi-point function secret sharing $\mathsf{MPFSS} = (\mathsf{MPFSS.Gen}, \mathsf{MPFSS.Eval}, \mathsf{MPFSS.FullEval})$.
- $G_{\mathsf{dual}}.\mathsf{Setup}(1^\lambda, \mathbb{F}, n, n', x)$: pick a random size-$t(\lambda)$ subset S of $[n']$, and a random vector $\vec{y} \xleftarrow{R} \mathbb{F}^t$. Let $s_1 < s_2 < \cdots < s_t$ denote the elements of S. Compute $(K_0, K_1) \xleftarrow{R} \mathsf{MPFSS.Gen}(1^\lambda, f_{S, x\vec{y}})$. Set $\mathsf{seed}_0 \leftarrow (\mathbb{F}, n, n', K_0, S, \vec{y})$ and $\mathsf{seed}_1 \leftarrow (\mathbb{F}, n, n', K_1, x)$. Output $(\mathsf{seed}_0, \mathsf{seed}_1)$.
- $G_{\mathsf{dual}}.\mathsf{Expand}(\sigma, \mathsf{seed}_\sigma)$.
  If $\sigma = 0$, parse seed_0 as $(\mathbb{F}, n, n', K_0, S, \vec{y})$. Set $\vec{\mu} \leftarrow \mathsf{spread}_n(S, \vec{y})$. Compute $\vec{v}_0 \leftarrow \mathsf{MPFSS.FullEval}(0, K_0)$. Output $(\vec{u}, \vec{v}) \leftarrow (\vec{\mu} \cdot H_{n',n}, -\vec{v}_0 \cdot H_{n',n})$.
  If $\sigma = 1$, parse seed_1 as $(\mathbb{F}, n, n', K_1, x)$. Compute $\vec{v}_1 \leftarrow \mathsf{MPFSS.FullEval}(1, K_1)$, and set $\vec{w} \leftarrow \vec{v}_1 \cdot H_{n',n}$. Output $\vec{w}$.

Figure 2: VOLE **Generator** G_{dual}**.**

3.3 Dual VOLE Generator

THEOREM 3.3. *Assuming that* $\mathsf{LPN}(n' - n, n', t/n')$ *holds over* $\mathbb{F}$ *with respect to the code with parity-check matrix* $H_{n',n}$ *and* MPFSS *is a secure multi-point function secret sharing,* G_{dual} *is a secure and correct* VOLE *generator.*

In the following, we prove Theorem 3.3.

3.3.1 Correctness. $\vec{u}x + \vec{v} = (\vec{\mu}x - \vec{v}_0) \cdot H_{n',n} = (\vec{\mu}x + \vec{v}_1 - \vec{\mu}x) \cdot H_{n',n} = \vec{v}_1 \cdot H_{n',n} = \vec{w}$.

3.3.2 Security. The first security requirement follows from the same argument as in the proof of Theorem 3.2. We now turn our attention to the second requirement.

Let $\mathcal{A}$ be a stateful PPT adversary, and let $(\mathbb{F}, n, n', x) \leftarrow \mathcal{A}(1^\lambda)$. We now prove the second security requirement. Consider the following game: compute $(\mathsf{seed}_0, \mathsf{seed}_1) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, n', x)$, set $(\vec{u}, \vec{v}) \leftarrow \mathsf{Expand}(0, \mathsf{seed}_0)$, and send $(\vec{u}, \vec{v}, \mathsf{seed}_1)$ to $\mathcal{A}$. Denote by β_0 the output of $\mathcal{A}$ in this game. Note that the input of $\mathcal{A}$ in this game is $\mathsf{seed}_1 = (\mathbb{F}, n, n', K_1, x)$, $\vec{u} = \vec{\mu} \cdot H_{n',n}$, and $\vec{v} = -\vec{v}_0 \cdot H_{n',n} = \vec{v}_1 \cdot H_{n',n} - \vec{\mu}x \cdot H_{n',n}$. Under the secrecy of the MPFSS, the key K_1 can be simulated solely from $(\mathbb{F}, n, n')$. It remains to show that the distribution of $(\vec{u}, \vec{v})$ is indistinguishable from the following distribution: pick $\vec{u'} \xleftarrow{R} \mathbb{F}^n$, set $\vec{v'} \leftarrow \mathsf{Expand}(1, \mathsf{seed}_1) - \vec{u'}x = \vec{v}_1 \cdot H_{n',n} - \vec{u'}x$, and output $(\vec{u'}, \vec{v'})$. To show it, it suffices to show that the distribution of $\vec{\mu} \cdot H_{n',n}$ is indistinguishable from the uniform distribution over $\mathbb{F}^n$.

Let $D_{n'-n,n} \in \mathbb{F}^{n'-n \times n'}$ be a generating matrix of the dual code of $H_{n',n}$ (*i.e.*, $D_{n'-n,n} \cdot H_{n',n} = 0^{n'-n \times n}$). Observe that for any vector $\vec{a} \in \mathbb{F}^{n'-n}$, it holds that $\vec{\mu} \cdot H_{n',n} = (\vec{\mu} + \vec{a} \cdot D_{n'-n,n}) \cdot H_{n',n}$. As $\vec{\mu}$ is a uniformly random noise vector with k non-zero coordinates over $\mathbb{F}^{n'}$ (given that the simulated K_1 is independent of $\vec{\mu}$), it holds that $\vec{\mu} + \vec{a} \cdot D_{n'-n,n}$ is indistinguishable from a uniformly random vector over $\vec{n'}$, under the $\mathsf{LPN}(n'-n, n, t/n')$ over $\mathbb{F}$ (using the fact that the dual matrix of a uniformly random matrix is itself a uniformly random matrix). Therefore, the distribution of $\vec{\mu} \cdot H_{n',n}$ is indistinguishable from the distribution obtained by picking $\vec{a'} \xleftarrow{R} \mathbb{F}^{n'}$ and outputting $\vec{a'} \cdot H_{n',n}$, which is exactly the uniform distribution over $\mathbb{F}^n$. This concludes the proof of security of G_{dual}.

3.3.3 Efficiency. Instantiating the MPFSS with the PRG-based construction outlined in Section 2.2.2, the setup algorithm of G_{dual} outputs seeds of size $t \cdot (\lceil \log n \rceil(\lambda + 2) + \lambda + \log_2 |\mathbb{F}|)$ bits, which amounts to $\tilde{O}(t)$ field elements over a large field ($\log_2 |\mathbb{F}| = O(\lambda)$). The Gaussian elimination attack on $\mathsf{LPN}(n'-n, n', t/n')$ takes time $O(1/(1 - t/n')^{n'-n}) \approx O(e^{(n'-n)\cdot t/n'})$ when t/n' is sufficiently small, and the ISD attack takes time $2^{f(n/n')\cdot t}$, where $f(n/n') \approx -\log_2(1 - n/n')$ when t/n' is sufficiently small [69]. This implies that this approach leads to a VOLE generator with arbitrary expansion factor; furthermore, taking n' to be a small multiple of n, e.g. $n' = 2n$, leads to a (conjectured) security of $O(e^t)$ which does not degrade with the expansion factor (and depends only on the seed size t). However, expanding the seed requires more work than for G_{primal}: it involves $t \cdot n'$ PRG evaluations and $O(n \cdot n') > n^2$ arithmetic operations.

Instantiating G_{dual} with parameters (t, n, n') over a field $\mathbb{F}$ yields a VOLE generator with seed length $t\cdot(\lceil \log n \rceil(\lambda+2)+\lambda+\log_2 |\mathbb{F}|)$ bits and output length $2n$ group elements (for $\mathsf{Expand}(0, \cdot)$) or n group elements (for $\mathsf{Expand}(1, \cdot)$). This VOLE generator is (T, ε)-secure iff $\mathsf{LPN}(n'-n, n', t/n')$ with code $D_{n'-n,n}$ is (T', ε)-secure and the MPFSS is (T'', ε)-secure, with $T' = T - O(n' \cdot (t\lambda + n \log_2 |\mathbb{F}|))$ and $T'' = T - O(n' \cdot n \cdot \log_2 |\mathbb{F}|)$.

3.4 Optimizations via Structured Matrices

We describe optimizations to the VOLE generators described so far. These optimizations allow us to obtain VOLE generators with *constant computational overhead.*

A downside of using both G_{primal} and G_{dual} with a random code is that this incurs quadratic computational complexity. Ideally, we would like to be able to compute $G_{\mathsf{primal}}.\mathsf{Expand}$ and $G_{\mathsf{dual}}.\mathsf{Expand}$ in time $O(n)$ (counted as a number of arithmetic operations and PRG evaluations).

Note that the complexity of $G_{\mathsf{primal}}.\mathsf{Expand}$ and $G_{\mathsf{dual}}.\mathsf{Expand}$ is dominated by multiplication by the matrix $C_{k,n}$ (or $H_{n',n}$) as well as evaluation of MPFSS.FullEval. In Section 4, we discuss optimization of MPFSS.FullEval. We now discuss an approach for decreasing the cost of the matrix-vector multiplication. These optimizations together allow us to reduce the computational complexity of both VOLE generators from quadratic to linear in the size parameter n.

Primal construction. A significant optimization of G_{primal} can be obtained by replacing the uniformly random matrix $C_{k,n}$ with a local linear code, where each column contains a small (constant) number of random non-zero coordinates. We note that using local alternatives to random linear encoding is relatively standard and is not known to weaken the security. Similar hardness conjectures were made in [2, 4]. Using such codes, computing $\vec{a} \cdot C_{k,n}$ for any vector $\vec{a}$ can be done using $O(n)$ arithmetic operations. Note that arithmetic pseudorandom generators with constant computational overhead can be obtained from the LPN assumption for some linear-time encodable code, see, e.g., [49]. This is needed for implementing the primal construction in linear time.

Dual construction. In the dual case, we need the matrix $H_{n',n}$ to define a compressive linear mapping, such that the code whose parity-check matrix is $H_{n',n}$ satisfies the LPN assumption. There are several alternative possibilities to implement this compressive mapping in linear time, which we outline below.

- One possibility is to use the transpose of the (randomized) linear-time encodeable code from [29]. As discussed in [29], LPN is a plausible assumption for these linear-time encodable codes as well as their dual codes. Moreover, the (compressive) transpose mapping can be computed with the same circuit complexity as the encoding (cf. [49]).
- Alternatively, one can replace the code from [29] by an LDPC code. The parity-check matrix of an LDPC code is a sparse matrix, for which LPN is conjectured to hold [2, 4]. Furthermore, while a naive encoding of an LDPC code requires quadratic time, recent results have established the existence of very efficient linear-time encoding algorithms for LDPC codes, both in the binary case [57] and in the general case, for codes over arbitrary fields [56]. The latter requires at most $n' \cdot \mathsf{rw}(D_{n'-n,n}) + \mathsf{w}(D_{n'-n,n})$ field multiplications, where $D_{n'-n,n}$ is the parity check matrix of $H_{n',n}$, $\mathsf{rw}(D_{n'-n,n})$ denotes the row-weight of $D_{n'-n,n}$, and $\mathsf{w}(D_{n'-n,n})$ denotes its total weight (i.e., the number of its non-zero elements); for $n' = O(n)$, this gives a linear time algorithm since $D_{n'-n,n}$ is sparse.
- Eventually, we observe that the only property we require from the encoding is to "sufficiently mix" the encoded vector: we do not require any structure or decoding properties. Hence, we conjecture that any suitable (linear-time) heuristic mixing strategy should work. A possibility is to apply a sequence of random atomic operations (switching two coordinates, multiplication by a constant, summing two coordinates). A better heuristic procedure (which achieves a better randomization with fewer steps) can be obtained using a mixing strategy based on expander graphs, such as the approach developped by Spielman in [66].

4 MPFSS CONSTRUCTIONS

An (n, t)-MPFSS for a multi-point function $f_{S,\vec{y}} : [n] \to \mathbb{G}$ can be readily constructed using t invocations to a DPF over $\mathbb{G}$:

- $\mathsf{MPFSS.Gen}(1^\lambda, f_{S,\vec{y}})$: denoting $s_1, \cdots, s_t$ (an arbitrary ordering of) the elements of S, for any $i \le t$, compute $(K_0^i, K_1^i) \xleftarrow{R} \mathsf{DPF.Gen}(1^\lambda, f_{s_i,y_i})$, where f_{s_i,y_i} is the point function over $\mathbb{G}$ which evaluates to y_i on s_i and to 0 otherwise. Output $(K_0, K_1) \leftarrow ((K_0^i)_{i\le t}, (K_1^i)_{i\le t})$.
- $\mathsf{MPFSS.Eval}(\sigma, K_\sigma, x)$: parse K_σ as $(K_\sigma^i)_{i\le t}$ and compute $z_\sigma \leftarrow \sum_{i=1}^{t} \mathsf{DPF.Eval}(\sigma, K_\sigma^i, x)$.

As with DPF, we can enhance an MPFSS with a full domain evaluation algorithm MPFSS.FullEval which, on input (σ, K_σ), outputs the vector $(\mathsf{MPFSS.Eval}(\sigma, K_\sigma, x))_{x\in[n]}$.

Plugging the construction of Theorem 2.2 leads to an (n, t)-MPFSS with key size $t \cdot (\lceil \log n \rceil (\lambda + 2) + \log_2 |\mathbb{G}|)$, where the computational cost of the evaluation algorithm is dominated by t group operations and $t \lceil \log n \rceil$ evaluations of a PRG, and the cost of a full domain evaluation is dominated by tn group operations and evaluations of a PRG.

4.1 Optimizing MPFSS Evaluation

The above simple reduction means that in MPFSS.FullEval the parties must make t passes over the entire domain $[n]$ for privately "writing" t entries (corresponding to the noisy coordinates) in a shared size-n vector. Below, we show how to improve this asymptotically, to writing a batch of t coordinates making a constant number of passes on the data. We discuss two alternatives: a concretely efficient approach which relies on a stronger (yet well-established) assumption than LPN, namely, the *regular syndrome decoding* assumption, and an asymptotically efficient approach using *batch codes* [48] which relies directly on LPN. Intuitively, the idea for the second approach is the following: evaluating MPFSS.FullEval on a vector shared between two parties can be seen as *writing t* entries (the noisy coordinates, known to the party who holds x) at secret locations (known to the other party), on a database secretly shared between the parties. A naive writing strategy makes t passes over the entire database, each pass writing a single entry at a secret position. Our goal, therefore, is to write a batch of t entries at secret positions using only a constant number of passes on the database.

A closely related problem involves secretly *reading* a batch of t secret entries from a database shared between several servers. This problem has been studied at length (see [48] and follow ups), and can be solved using a combinatorial object called *batch code*. Our solution essentially applies the same strategy, formulating the task as a private writing problem, and shows that the same batch-code-based strategy can similarly be used for this related task.

4.1.1 Optimized MPFSS Evaluation using Regular Syndrome Decoding (RSD). The RSD assumption is a strengthening of the LPN assumption which was introduced in [8] as the assumption underlying the security of a candidate for the SHA-3 competition, and which has been studied at length (see [45] for a recent survey about the cryptanalysis of the RSD assumption and a detailed discussion about its security). It states that LPN remains hard, even if the sparse noise vector is *regular*, meaning that it is divided into t blocks of size n/t each, each block containing a single random 1, and zeroes everywhere else. Furthermore, there is a smooth tradeoff between the underlying assumption (from LPN to RSD) and the complexity (from tn to n operations): one can consider overlapping subsets instead of disjoint subsets, with larger subsets leading to a longer MPFSS evaluation time but a noise pattern closer to uniform (hence an assumption closest to plain LPN).

While the noise distribution obtained with this procedure is not uniform anymore, it seems to resist all known attacks [45]. In particular, note that it is not broken by the attack of [7], (which, in particular, does not apply when we use random large enough overlapping subset instead of small non-overlapping subsets): the attack of [7] requires at least a quadratic number of samples (note that for G_{dual}, the number of samples is $N + o(N)$, where $N = n' - n$ is the dimension).

Using a regular noise pattern instead of a random noise pattern directly allows to reduce MPFSS.FullEval to t calls to a DPF on length-n/t vectors, for a total cost of n operations in the underlying field $\mathbb{F}$ and at most $n(1 + \lceil \log |\mathbb{F}|/(\lambda + 2) \rceil)$ PRG evaluations [18]. However, this comes at the cost of relying on the stronger RSD assumption; below, we outline an alternative strategy which also leads to an $O(n)$ cost, without relying on RSD.

4.1.2 Batch Codes. We first recall the definition of batch codes, from [48].

Definition 4.1 (Batch Code [48]). An (n, N, t, m)-batch code over an alphabet Σ encodes any string $x \in \Sigma^n$ into an m-tuple of strings $(z_1, \cdots, z_m) \in \Sigma^*$ (called *buckets*) of total length N, such that any t-tuple of coordinates of x can be recovered by reading at most a single entry from each bucket.

Specifically, we will rely on a *combinatorial batch code* (CBC) [48, 68], a special type of batch code in which an encoding of a string x consists only of replicating the coordinates of x over "buckets" (*i.e.*, each bucket contains a subset of the coordinates of x).

A CBC can be represented by a bipartite graph, with n left-nodes, m right-nodes, and N edges. Each string z_j, $j \in [m]$ corresponds to the j-th right-node, where the value of z_j is set to the concatenation of (x_i) for $i \in [n]$ such that (i, j) is an edge (with some canonical ordering). The CBC requirement states that any subset of t left-nodes has a matching to the m right nodes. By Hall's theorem, such a bipartite graph represents an (n, N, t, m)-CBC if and only if it satisfies the following weak expansion property: each subset S of at most t left-nodes has at least $|S|$ neighbors on the right.

4.1.3 From CBC to Better MPFSS. Assume for now that, for given parameters t and $n = O(t^s)$ (for some constant expansion factor s), there is a $(n, N = O(n), t, m = t^{1+\varepsilon})$-CBC (for some constant $\varepsilon > 0$).

Loosely speaking we use such a batch code to construct an efficient MPFSS.FullEval by the following steps. Instead of t instances of DPF with domain size n, we will use m DPF instances, each with domain size $|z_j|$ (for $j \in [m]$). Namely, the multi-point function over $[n]$ maps $n - t$ inputs to 0 and t values to group elements. Concatenating these n values together we obtain a string x which can be batch-encoded into m strings $z_1, \ldots, z_m$ with total length N. By the property of batch codes the t points defined by the multi-point function can be recovered by reading one entry of each of the m strings. Therefore, MPFSS.FullEval can be implemented by running DPF.FullEval m times, with the domain size of the j-th invocation corresponding to the length of z_j for a total length of $O(N)$ (instead of total length tn in the simple reduction of MPFSS.FullEval to DPF.FullEval). The details follow.

Let $T_1, \cdots, T_m \subset [n]$ denote the left neighbors of each right-node of the graph associated to the CBC. Let $f_{S,\vec{y}} : [n] \to \mathbb{F}$ be a t-point function, with $S = \{s_1, \cdots, s_t\}$. Let DPF = (DPF.Gen, DPF.Eval, DPF.FullEval) be a function secret sharing for the class of all point functions from $|z_j|$ to $\mathbb{F}$.

- MPFSS.Gen$(1^\lambda, f_{S,\vec{y}})$: let $I = \{i_1, \cdots, i_m\}$ denote a size-m subset of $[n]$ such that $i_j \in T_j$ for any $j \leq m$, and $S \subset I$ (such a subset necessarily exists by definition of a CBC). For $j = 1$ to m, define $f_j : [|z_j|] \to \mathbb{F}$ to be the following function: if there exists ℓ such that $s_\ell = i_j$, f_j is the point function

that outputs y_ℓ on i_j, and 0 otherwise. Else, f_j is the all-zero function, which is a point function with a 0 value defined for the designated point. Compute $(K_0^j, K_1^j) \xleftarrow{R} \mathsf{DPF.Gen}(1^\lambda, f_j)$. Output $(K_0, K_1) \leftarrow ((K_0^j)_{i \le m}, (K_1^j)_{i \le m})$.
- $\mathsf{MPFSS.FullEval}(\sigma, K_\sigma)$: parse K_σ as $(K_\sigma^j)_{i \le m}$. Compute $\vec{\alpha}$ by $\vec{\alpha} = \sum_{j=1}^m \mathsf{DPF.FullEval}(\sigma, K_\sigma^j)$. Output $\vec{\alpha}$.

The correctness of the above construction immediately follows from the CBC property. Regarding efficiency, a key that MPFSS.Gen outputs is slightly longer compared to the simple construction outlined in the beginning of this section (the length is $O(t(\lambda\lceil\log n\rceil + \log|G|))$ in the simple construction and $O(t^{1+\varepsilon}(\lambda\lceil\log n/t\rceil + \log|G|))$ in the batch-code based construction). However, the computational cost of the simple construction is dominated by $O(tn)$ PRG evaluations while the batch-code based method requires $O(\sum_{j=1}^m |z_i|) = O(n)$ PRG evaluations saving a factor of $O(t)$ in computation.

4.1.4 Instantiating CBC. Unfortunately, known explicit constructions of (provable) expander graphs fail to match our efficiency requirements. We outline below two standard ways of getting around this issue.

- First, consider a random construction of the graph, as follows: pick any constant ε, set $d \leftarrow (1+s)\cdot\varepsilon + 1$, and $m \leftarrow t^{1+\varepsilon}$. For each left-node u, repeat the following d times: pick a uniformly random right-node v, and add the edge (u, v) to the graph if it does not already exist. By a standard union bound, with probability at least $1 - t^{-2(d-1)}$, the graph will satisfy the required expansion property. Note that this is a one-time setup, which fails with a probability $1/t^{\Omega(d)}$ that can be made as small as we want, and which is independent of both the running time of any adversary, and the number of executions of the MPFSS algorithms.
- Second, one can consider a heuristic approach using some fixed sequence of bits (say, e.g., the digits of π) and interpreting it as the graph of a (n, N, k, m)-CBC under some fixed translation. Assuming that this heuristic leads to a graph with the required expansion property can be viewed as a relatively weak combinatorial assumption, which we refer to as the existence of *explicit polynomially unbalanced bipartite expanders*. This assumption has been made (either explicitly or implicitly) in prior works on expander-based cryptography [3, 4, 6, 41, 49].

Indeed, in the context of this work, this issue is in fact even less of a concern. Observe that if the graph of the CBC fails to be sufficiently expanding then the noise distribution will slightly deviate from being uniform. However, the LPN assumption for such slightly skewed noise distributions remains a very conservative assumption. Therefore, we get the following guarantee: either a simple combinatorial assumption holds, and our VOLE generators are secure under the standard LPN assumption; or it fails, in which case our VOLE generators remain secure assuming a plausible variant of LPN.

5 EFFICIENCY OF VOLE GENERATION

In this section, we discuss the asymptotic and concrete efficiency we can obtain with the VOLE generators G_{primal} and G_{dual}.

We start with asymptotic efficiency. Using an "LPN-friendly" code which is linear-time encodable (alternatively, its dual is linear-time encodable for the dual construction), and using the CBC-based MPFSS (alternatively, using the "regular noise" variant of LPN, as in Section 4.1.1) our VOLE generators can be computed using $O(n)$ arithmetic operations. This is captured by the following theorem.

Theorem 5.1. *Assume the existence of explicit constant-degree polynomially unbalanced bipartite expanders (see Section 4.1.4). Then the following holds.*

- **Primal.** *For any $\varepsilon > 0$ and $1 < c < 2$, under the $\mathsf{LPN}(n^{1/c}, n, n^{\varepsilon - 1/c})$ assumption over $\mathbb{F}$ with respect to a linear-time encodable code, there exists a VOLE generator G_primal over $\mathbb{F}$ with seed length $n^{1/c}$ field elements and output length n.*
- **Dual.** *For any $\varepsilon > 0$ and $d > 1$, under the $\mathsf{LPN}(n/2, n, n^{\varepsilon - 1/d})$ assumption over $\mathbb{F}$ with respect to a code whose dual H is linear-time encodable, there exists a VOLE generator G_{dual} over $\mathbb{F}$ with seed length $n^{1/d}$ field elements and output length n.*

In both cases, computation of G requires $O(n)$ field operations. Furthermore, using the regular syndrome decoding assumption instead of LPN (with the same parameters) removes the need for explicit expanders.

We note that the random local encoding of Alekhnovich or the code ensemble from [29] (see [4] and Section 3.4) can be used to instantiate the linear-time LPN assumption.

5.1 Optimal Seed Size for a Given Output Size

We turn to analyze the concrete efficiency of our VOLE generators, starting with a concrete optimization of the seed size. The optimal seed size for a given desired output size can be obtained by numerically solving an optimization problem in two variables (k and t) under the constraint that the corresponding LPN instance requires 2^σ bit operations to be solved. We represent on Table 1 the optimal choices of parameters to minimize the size of the seed for a given output size, under the constraint that the corresponding LPN problem requires 2^{80} arithmetic operations to be solved with a Gaussian elimination attack (which is the best known attack as of today on LPN for this regime of parameters).

G_{dual} satisfies the counterintuitive property that the optimal seed size become *smaller* as the target output size grows larger. This is a consequence of the fact that $\mathsf{LPN}(n'-n, n', t/n')$ becomes harder to attack as n grows, independently of the other parameters (unlike the LPN instance which underlies G_{primal}, $\mathsf{LPN}(k, n, t/n)$): the cost of breaking $\mathsf{LPN}(n'-n, n', t/n')$ is roughly $e^t \cdot (n'-n)^w$, where $w \approx 2.7$ is a conservative estimated exponent of practical matrix multiplication algorithms (we further conservatively assume the multiplicative constant to be 1 in the matrix multiplication algorithm). Therefore, to get the same level of security, the optimal choice of t is smaller when n, n' increase. However, this comes at the cost of a higher computation: recall that expanding the seed of G_{dual} requires $O(n \cdot n')$ arithmetic operations. Nonetheless, using a linear-time encodable code (see Section 3.4) can bring this cost down to $O(n') = O(n)$ for the same (conjectured) security.

Table 1: Optimal parameters of G_{primal} and G_{dual} for a given output size. Both the security parameter λ and the bitsize of field elements $\log_2|\mathbb{F}|$ are set to 128. We set n' to $2n$. The parameters are optimized under the constraint that solving the corresponding LPN instance must require at least 2^{80} arithmetic operations with either Gaussian elimination or ISD. The seed size is counted as a number of field elements (bitsize divided by 128) to facilitate comparison with the trivial solution (directly sharing the output vector-OLE). Ratio is n divided by the seed size; it measures the gain in storage with respect to the trivial solution.

G_{primal}

n	2^{11}	2^{14}	2^{16}	2^{18}	2^{20}	2^{24}
t	74	204	329	600	1114	3613
k	984	2659	6167	12664	25460	106933
Seed size	1959	5968	12171	24833	50316	202226
Ratio	1.05	2.74	5.38	10.56	20.84	82.96

G_{dual}

n	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
k	71	64	57	49	41	34
Seed size	863	908	924	894	832	759
Ratio	1.19	4.51	17.73	73.31	315.08	1382

5.2 Heuristic Optimizations

In this section, we describe heuristic optimizations which strongly improve the computational efficiency of G_{primal} and G_{dual}, while seemingly resisting known attacks.

5.2.1 Better Encoding. While using uniformly random matrices $C_{k,n}$ and $H_{n',n}$ in G_{primal} and G_{dual} allows to reduce their security to the hardness of the standard LPN assumption, it is wasteful in terms of computation. Heuristically, it suffices to apply a linear mapping that "sufficiently mixes" the coordinates of the encoded vector (which is either some random small vector in G_{primal}, or some long noise vector in G_{dual}) to defeat Gaussian elimination attacks.

Concretely, we suggest the following heuristic procedure:

- **For** G_{primal}: set $m \leftarrow n$. Pad $\vec{a}$ with zeroes to get a size-m vector. Apply $10m$ rounds of one of the following three operations, picked at random (and fixed in the specifications of G_{primal}):
  - Swap two coordinates
  - Multiply a coordinate by a random scalar
  - Add to one of the coordinates a multiple of another

  where the coordinates on which to apply the operations are picked at random (and fixed in the specifications).
- **For** G_{dual}: set $m \leftarrow n' = 2n$. Apply $10m$ rounds of the same three operations as above. Truncate the output to length-n.

The cost of applying the above heuristic linear mapping is dominated by $\approx 7m$ multiplications over $\mathbb{F}$.

5.2.2 Simplified Full Domain Evaluation. We described in Section 4.1 a strategy to optimize the full evaluation procedure of the MPFSS. Using, e.g., the RSD-based solution, the entire cost of MPFSS.FullEval is $2m$ PRG evaluations for field sizes that are roughly the size of the security parameter.

5.2.3 Time Complexity Estimates. The computational performance of our VOLE generators is dominated by computing a linear mapping of the seed over a field $\mathbb{F}$ and by computing MPFSS.FullEval. Emmart et al. [30] report 12.2 billion modular multiplications per second over a field $\mathbb{F}_p$ for a 128-bit prime p using a common graphics card (Nvidia GTX 980 Ti). Hence, applying the heuristic linear mapping can be performed in at most $7m/(12.2 \cdot 10^9)$ seconds on a personal computer with the appropriate graphics card.

Implementing the PRG using AES,[2] each PRG evaluation amounts to 2 calls to AES. A computer equipped with an Intel i7-6700 can encrypt 2607 megabytes per second using AES-128-GCM.[3] Therefore, a computer equipped with the same processor can execute a heuristically optimized full domain evaluation, dominated by $4m$ AES encryption operations in $m/(41 \cdot 10^6)$ seconds.

As an example for this estimate, consider VOLE output size of 2^{20} field elements for a prime field with a 128-bit prime. In the primal generator, the linear mapping part requires $7 \cdot 2^{20}/12.2 \cdot 10^9 \approx 0.6$ milliseconds. In the same setting, the MPFSS scheme uses $4 \cdot 2^{20}$ AES operations which require 25.5 milliseconds. Taken together the time for the primal VOLE generation is approximately 26.1 milliseconds. For the dual generator with the same n and $\mathbb{F}$ we have that $m = 2^{21}$ and therefore the linear mapping takes 1.2 ms, MPFSS.FullEval takes 51 ms and the total is 52.2 ms.

Note that for a smaller field size, e.g. a prime field of length 64 bits, the MPFSS.FullEval is about twice as fast, using the "early termination" optimization of [18]. This optimization results (for $\lambda = 127$ and $|\mathbb{F}| \leq 2^{64}$) in halving the time of MPFSS.FullEval and therefore requiring about 13 ms for the primal generator.

We stress that the numbers reported in this section assume a heuristic encoding based on an arbitrary constant (here, 7), which might be overly optimistic or overly pessimistic, and deserves further investigations. Therefore, the numbers should only be seen as an estimation of a potentially achievable efficiency (a few dozen milliseconds on practical instances) rather than an exact efficiency statement. Note also that with the configuration above, the cost as a function of the constant c (which was set to 7 above) and $c' = n'/n$ (which was arbitrarily set to 2; smaller values give a more efficient computation for G_{dual}, but require a longer seed to achieve 80 bits of LPN security) is $0.085 \cdot c + 25.5$ millisecond for G_{primal}, and $c' \cdot (0.085 \cdot c + 51)$ millisecond for G_{dual}. Therefore, one can easily increase c to get a larger security margin without significantly increasing the cost.

If one wants to ressort instead on a more conservative assumption, an alternative is to use Alekhnovich's assumption [2]. In the case of G_{primal}, this would amount to $c \cdot m$ multiplications, where c is the row-weight of the code matrix (which is a constant in

[2] The PRG can either be defined to use AES in counter mode, i.e. PRG(s) is $\text{AES}_{s||0}(0)$, $\text{AES}_{s||0}(1)$ for a seed $s \in \{0,1\}^{127}$ or a fixed key alternative $\text{AES}_{k_0}(s||0) \oplus s||0$, $\text{AES}_{k_1}(s||0) \oplus s||0$ for fixed keys k_0, k_1. The choice of AES is motivated by the hardware support for AES encryption and decryption in modern CPUs.

[3] https://calomel.org/aesni_ssl_performance.html

Alekhnovich's assumption), the same as with the heuristic encoding described above. In the case of G_{dual}, using the linear-time encoding algorithm of [56], which requires at most $c \cdot (2m - n)$ (since it is bounded by $n' \cdot \text{rw}(D_{n'-n,n}) + \text{w}(D_{n'-n,n})$ and $m = n'$, $\text{rw}(D_{n'-n,n}) = c$, see Section 3.4, and $\text{w}(D_{n'-n,n}) < (n' - m) \cdot \text{rw}(D_{n'-n,n}) = c \cdot (m - n)$). Using for example $m = n' = 2n$ gives a cost of $3 \cdot c \cdot m$ multiplications, three times larger than the cost obtained with the heuristic encoding.

5.3 Distributed Generation of MPFSS

So far we thought of the VOLE generator Setup as being performed by a trusted dealer who samples and sends seed_0 and seed_1 to the respective parties. In practice, the trusted dealer can be emulated via secure two-party computation. For both of our VOLE generator constructions, the complexity of Setup is dominated by the execution of MPFSS.Gen which in turn consists of a series of executions of DPF.Gen. More specifically, for each DPF.Gen, one party (VOLE sender) selects and knows the *position* of the the designated DPF point and the *evaluation* of the DPF is taken to be the product of the noise value y_i known to the VOLE sender and the secret x known to the second party (VOLE receiver). Note that this is also the case for the batch-code based and RSD based constructions of MPFSS.

In the DPF.Gen construction of [18] for point functions over the domain $\mathbb{F}^n$ the two output keys are $K_0 = (s_0^{(0)}, cw_1, \ldots, cw_{\nu+1})$ and $K_1 = (s_1^{(0)}, cw_1, \ldots, cw_{\nu+1})$ where $s_0^{(0)}, s_1^{(0)}$ are two random seeds for the PRG and $\nu = \min\{\lceil \log n - \log \frac{\lambda}{\log |\mathbb{F}|} \rceil, \log n\}$. Gen proceeds in $\nu + 1$ steps. In the i-th step it expands $s_0^{(i-1)}$ and $s_1^{(i-1)}$ by using one PRG invocation for each seed and obtains $s_0^{(i)}, s_1^{(i)}$ and cw_i. In the final step the algorithm computes $cw_{\nu+1}$ as a function of the expanded seeds and the target value. We discuss and analyze two different approaches for distributing DPF.Gen.

5.3.1 Generic 2PC. Any protocol for 2PC can compute the output of Gen securely. Both the communication and computation of the protocol are dominated by two factors: λ OTs for one seed and $2(\nu + \mu)$ secure evaluations of the PRG for $\mu = \lceil \frac{\log |\mathbb{G}|}{\lambda + 2} \rceil$. Setting $\lambda = 127$ and the PRG to two AES evaluations, as suggested previously, results in 127 OTs and $4(\nu + \mu)$ secure evaluations of AES.

Assume that securely evaluating AES is implemented by an efficient protocol such as [64] or [70]. Wang et al. [70] uses an Amazon EC2 c4.8xlarge instance over a LAN, with statistical security parameter $\rho = 2^{-40}$, and securely evaluates a single AES instance in 16.6 milliseconds, while the amortized cost of 1024 AES evaluations is roughly 6.66 milliseconds in the malicious model (and 2.1 milliseconds in the semi-honest model). The evaluation of the OTs required for the AES is about 20 milliseconds. Assuming that the amortized cost of an AES evaluation for our likely range of parameters, i.e. several dozen AES evaluations, is about 10 milliseconds in the malicious setting implies that the total execution time of the protocol is about $40(\nu + \mu) + 20$ milliseconds. The communication complexity is roughly $(\nu + \mu)(\frac{|C|\rho}{\log |C| + \log(\nu+\mu)})$ field elements, where $|C| = 6800$ is the circuit size, or more precisely the number of AND gates in the evaluated AES circuit. For example, if $n = 2^{20}$ and $|\mathbb{F}| \leq 2^{128}$ then by Table 1 the number of times that DPF.Gen is executed in the dual generator is $k = 34$. Therefore, the running time is estimated to be $34 \cdot 840$ ms, or 29.2 seconds and the communication is about 11.34 million field elements.

These numbers can be further improved using an MPC-friendly PRG with few AND gates instead of AES; e.g., using LowMC [1] would give approximately a 23-time improvement for communication and computation of the setup.

5.3.2 Black-Box Approach. The most expensive part of using generic 2PC to distribute DPF.Gen is the multiple evaluations of AES. An alternative approach which treats the PRG as a black box was offered by Doerner and shelat in [27]. The idea is to expand all the seeds at each level i of the tree described by DPF.FullEval. In each such level the only difference between the expanded strings computed by each party are the result of the two expanded seeds along the path to the designated point. Securely computing cw_i is possible using a single OT and local computation that is proportional to the size of the tree level. The total computation time of this protocol is dominated by $\nu + 1$ OTs and $2^{\nu+1}$ locally computed AES operations, which for $n = 2^{20}$ $|\mathbb{F}| \leq 2^{128}$ is 2^{21} AES operations. Using the previous estimate of 2607 MBPS for AES on a standard PC we get that the computation requires only 13 ms and that the number of oblivious transfers is much lower than what is required for the competing 2PC approach (21 OTs vs. 127 OTs). Therefore, for $n = 2^{20}$ using the black box approach to distribute the setup is much cheaper than the generic 2PC approach.

6 APPLICATIONS

As discussed in the Introduction, VOLE generators can be used as a general-purpose tool in any application that benefits from large VOLE instances.

6.1 Secure Arithmetic Computation

There are numerous applications of secure computation that benefit from representing the function being computed as an arithmetic circuit. See, e.g., [4, 28, 52, 53, 61] and reference therein. Many of these applications involve multiplying a secret scalar by a secret vector, where the two inputs can either be held by a single party of secret-shared by the two parties. Such scalar-vector multiplication is a useful building block for more complex protocols that involve matrix-vector or matrix-matrix multiplication.

More concretely, suppose that a scalar $x \in \mathbb{F}$ and vector $\vec{u} \in \mathbb{F}^n$ are additively shared between P_0 and P_1. Let x_0, x_1 and $\vec{u}_0, \vec{u}_1$ denote the shares. Then, an additive sharing of $x \cdot \vec{u}$ can be obtained via two invocations of VOLE, by breaking the product $(x_0 + x_1)(\vec{u}_0 + \vec{u}_1)$ into four terms and using the two VOLE instances to obtain additive shares of the cross-terms $x_0 \cdot \vec{u}_1$ and $x_1 \cdot \vec{u}_0$ (the other two terms can be computed locally). Other than being directly useful for secure linear algebra, this sub-protocol can be used to speed up protocols for arithmetic circuits that have a large multiplication fan-out.

6.1.1 Vector OLE *from Pseudorandom* VOLE *Generator.* We now describe and analyze the standard method for converting random VOLE into standard VOLE (cf. [52]), and prove its security when using the output of the VOLE generator to produce a random VOLE. This justifies the security notion of VOLE generators we put forward in Definition 3.1.

We start by recalling the standard protocol for implementing VOLE from an ideal random VOLE correlation.

Preprocessing. A trusted dealer picks $(\vec{r_u}, \vec{r_v}, r_x) \overset{R}{\leftarrow} \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}$, sets $\vec{r_w} \leftarrow \vec{r_u} r_x + \vec{r_v}$, and outputs $(\vec{r_u}, \vec{r_v})$ to P_0 and $(\vec{r_w}, r_x)$ to P_1.

Input. P_0 has input $(\vec{u}, \vec{v})$, and P_1 has input x.

Protocol. P_1 sends $m_x \leftarrow x - r_x$. P_0 sends $\vec{m_u} \leftarrow \vec{u} - \vec{r_u}$ and $\vec{m_v} \leftarrow m_x \vec{r_u} + \vec{v} - \vec{r_v}$. P_1 outputs $\vec{w} \leftarrow \vec{m_u} x + \vec{m_v} + \vec{r_w}$.

Correctness: $\vec{w} = \vec{m_u} x + \vec{m_v} + \vec{r_w} = (\vec{u} - \vec{r_u})x + (x - r_x)\vec{r_u} + \vec{v} - \vec{r_v} + \vec{r_u} r_x + \vec{r_v} = \vec{u}x + \vec{v}$. Security is straightforward.

We now consider a modification of the above protocol that replaces the ideal random VOLE correlation by the output of the VOLE generator:

Preprocessing. A trusted dealer picks $r_x \overset{R}{\leftarrow} \mathbb{F}$, proceeds to compute $(\mathsf{seed}_0, \mathsf{seed}_1) \overset{R}{\leftarrow} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, r_x)$, and outputs seed_0 to P_0 and (r_x, seed_1) to P_1.

Offline Expansion. P_0 computes $(\vec{r_u}, \vec{r_v}) \leftarrow \mathsf{Expand}(0, \mathsf{seed}_0)$. P_1 computes $\vec{r_w} \leftarrow \mathsf{Expand}(1, \mathsf{seed}_1)$.

Input. P_0 has input $(\vec{u}, \vec{v})$, and P_1 has input x.

Protocol Π_{VOLE}. P_1 sends $m_x \leftarrow x - r_x$. P_0 sends $\vec{m_u} \leftarrow \vec{u} - \vec{r_u}$ and $\vec{m_v} \leftarrow m_x \vec{r_u} + \vec{v} - \vec{r_v}$. P_1 outputs $\vec{w} \leftarrow \vec{m_u} x + \vec{m_v} + \vec{r_w}$.

Correctness follows from the correctness of the VOLE generator and the same analysis as before.

Proposition 6.1. *Assuming* (Setup, Expand) *is a secure* VOLE *generator (as in Definition 3.1), the protocol* Π_{VOLE} *is a secure vector-*OLE *protocol in the preprocessing model.*

Proof. We exhibit a simulator Sim that generates a view indistinguishable from an honest run of the protocol as long as a single party is corrupted.

Case 1: P_0 is corrupted. In the preprocessing phase, Sim picks a random $r_x \overset{R}{\leftarrow} \mathbb{F}$, computes $(\mathsf{seed}_0, \mathsf{seed}_1) \overset{R}{\leftarrow} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, r_x)$, and outputs seed_0 to P_0. In the online phase, Sim sends $m_x \overset{R}{\leftarrow} \mathbb{F}$. Observe that the view of P_0 in this simulated protocol is perfectly equivalent to an honest run of the protocol where P_1 would pick a uniformly random r'_x and send $m_x \leftarrow x - r'_x$ instead of computing $m_x \leftarrow x - r_x$ using the random r_x received from the trusted dealer. This implies that distinguishing the simulated protocol from the real one is equivalent to distinguishing a run of the protocol with the random r_x picked by the dealer from a run of the protocol with a fresh random r'_x. Therefore, the indistinguishability between the simulated protocol and the real protocol follows immediately from the first security requirement of the VOLE generator.

Case 2: P_1 is corrupted. In the preprocessing phase, Sim picks a random $r_x \overset{R}{\leftarrow} \mathbb{F}$, computes $(\mathsf{seed}_0, \mathsf{seed}_1) \overset{R}{\leftarrow} \mathsf{Setup}(1^\lambda, \mathbb{F}, n, r_x)$, and outputs (r_x, seed_1) to P_1. In the online phase, Sim receives m_x from P_1, and the target output $\vec{w}$ of P_1. Sim computes $\vec{r_w} \leftarrow \mathsf{Expand}(1, \mathsf{seed}_1)$, and sets $\vec{m_w} \leftarrow \vec{w} - \vec{r_w}$ and $x \leftarrow m_x + r_x$. Sim picks $\vec{m_u} \overset{R}{\leftarrow} \mathbb{F}^n$ and set $\vec{m_v} \leftarrow \vec{m_w} - \vec{m_v} x$. Sim sends $(\vec{m_u}, \vec{m_v})$ to P_1. The indistinguishability between the simulated protocol and the real protocol follows immediately from the second security requirement of the VOLE generator. □

6.1.2 Malicious Security. An attractive feature of Π_{VOLE} is that as long as the preprocessing is trusted then Π_{VOLE} is secure against a malicious adversary. The reason is that if one of the players is corrupt then any deviation it makes from the protocol can be simulated by a corresponding change of input in the ideal model. This effectively means that our VOLE generator can be used as a plug-and-play alternative to ideal VOLE, as long as the setup implementation is secure (e.g., it is distributed between the parties using maliciously secure two-party computation).

In more detail, if P_1 is corrupted then since the only message it sends in the protocol is $m_x = x - r_x$ its only possible deviation is to change that message to some m'_x. The trusted setup outputs r_x and therefore an honest player would send the message m'_x on input $x' = m'_x + r_x$ and output $\vec{w}' = \vec{u}x' + \vec{v}$. As a consequence the simulator for P_1 with input x' in the semi-honest setting simulates the malicious adversary with input x, which proves that in this case the protocol is secure in the malicious setting.

If P_0 is corrupted then it can only output two messages $\vec{m'_u}$ and $\vec{m'_v}$ that are different from the real vectors. An honest player would send $\vec{m'_u}$ on input $\vec{u'} = \vec{m'_u} + \vec{r_u}$ and $\vec{m'_v}$ on input $\vec{v'} = \vec{m'_v} - \leftarrow m_x \vec{r_u} + \vec{r_v}$ and the output would be $\vec{w'} = \vec{u'}x + \vec{v'}$. Again there exists a simulator for a malicious adversary since a simulator exists in the semi-honest case with inputs $\vec{u'}$ and $\vec{v'}$.

6.1.3 Rate-1/2 VOLE *protocol in the plain model.* By distributing the setup of our (primal or dual) VOLE generators using general-purpose protocols for secure two-party computation, we get VOLE protocols in the plain model with attractive efficiency features. The protocols can be implemented in a constant number of rounds and have asymptotic communication rate of 1/2. That is, the communication complexity is dominated by the cost of communicating two vectors in $\mathbb{F}^n$. Using the dual construction, the protocol can be based on OT together with LPN with a linear number of samples $n = O(k)$ (in fact, $n = k + o(k)$ samples suffice) and a slightly sub-linear noise ($n^{1-\epsilon}$ noisy samples). This is strictly better than the flavor of LPN known to imply public-key encryption [2].

Combined with linear-time encodable LPN-friendly codes, we get VOLE protocols in the plain model that have constant computational overhead and make a black-box use of the underlying field. Compared to the recent constant-overhead VOLE protocols from [4], the protocol Π_{VOLE} obtained by combining Proposition 6.1 and Theorem 5.1 has the qualitative advantage of non-interactive generation and the quantitative advantage of asymptotic rate of 1/2 (compared to 1/3 in [4]). The underlying LPN assumption is similar but technically incomparable: our protocol requires LPN with a slightly sub-constant noise rate (compared to constant noise rate in [4]) but also uses a smaller number of samples (linear vs. super-quadratic). Another advantage of our protocol is that it avoids any kind of erasure decoding or Gaussian elimination that were required in [4] and in other previous protocols. Finally, a unique feature of our protocol is that it can achieve security against malicious parties at a vanishing amortized cost.

Focusing on communication complexity alone, VOLE with rate 1 could be previously obtained via the Damgård-Jurik encryption scheme, and rate 1/2 could be obtained from LWE, DDH, or Paillier via homomorphic secret sharing [16, 19, 26, 32]. Note that since neither our flavor of LPN nor OT are known to imply collision-resistant hashing (CRH), rate 1/2 seems to be a barrier under these assumptions. Indeed, using the techniques of [46] one can show that any constant-round (semi-honest) VOLE protocol that achieves better than 1/2 rate implies *constant-round* statistically hiding commitment, which currently can only be based on CRH.

6.2 Non-Interactive Zero-Knowledge with Reusable Correlated Randomness Setup

Consider the following model for non-interactive zero-knowledge (NIZK) with setup. In an offline phase, before the statements to be proved are known, the prover and the verifier receive correlated randomness from a trusted dealer. Alternatively, they may generate this correlated randomness on their own using an interactive secure computation protocol that is carried out once and for all during a preprocessing phase. Then, in the online phase, the prover can prove each NP-statement *non-interactively*, by sending a single message to the verifier.

We would like the setup to be *reusable* in the sense that the number of statements that can be proved is polynomially larger than the communication cost of the setup. Moreover, the soundness of the protocol should hold even if the prover can learn whether the verifier accepts or rejects a maliciously generated proof. NIZK protocols based on OT (e.g., [50, 55]) fail to satisfy this property, since the prover can gradually learn the verifier's OT selections via small perturbations of an honest prover's strategy.

We observe that a suitable type of zero-knowledge linear PCPs for NP, which exist unconditionally, can be compiled in a simple way into information-theoretic reusable NIZK protocols in the VOLE-hybrid model. Concretely, proving n instances of satisfiability of an arithmetic circuit of size s over $\mathbb{F}$ requires $O(s)$ instances of VOLE of length $O(n)$ each, where the verifier's VOLE inputs are assumed to be honestly generated. (This is a simplified version of a similar construction from [21] which is zero-knowledge against a malicious verifier.) Applying our VOLE generator, the cost of the setup depends only on s and not on n, and each circuit satisfiability instance consumes only a constant number of entries from each of the $O(s)$ VOLE instances.

Following the local expansion of the VOLE seeds, which does not require interaction, generating and verifying each proof involves only $O(s)$ field operations on both sides (and no "cryptographic" computations), and the proof consists of $O(s)$ elements of $\mathbb{F}$. This should be contrasted with traditional approaches to SNARGs, which can have sublinear communication[4] and verifier computation, but on the other hand are much heavier in terms of prover computation. Our NIZK constructions are particularly attractive in settings where the prover and verifier have comparable computational resources and where communication is relatively cheap.

We now define the notion of linear proof systems on which we rely, which is a variant of the "linear interactive proof" model from [14]. At a high level, such a proof system proceeds by multiplying a proof matrix Π picked by the prover by an *independently chosen* query vector $\vec{q}$ picked by the verifier, where the verifier decides whether to accept or reject based on $\vec{q} \cdot \Pi$ alone. Note that unconditional zero-knowledge is possible in this model because of the restricted mode of interaction. We will later use a VOLE generator to securely realize such proofs non-interactively with reusable setup.

Definition 6.2 (HVZK-LIP). An *honest-verifier zero-knowledge linear interactive proof* (HVZK-LIP) is a triple of algorithms (Prove, Query, Verify) with the following syntax:

- Prove$(\mathbb{F}, C, x, w)$ is a PPT algorithm that given an arithmetic verification circuit $C : \mathbb{F}^\ell \times \mathbb{F}^L \to \mathbb{F}$, an input (NP-statement) $x \in \mathbb{F}^\ell$, and witness $w \in \mathbb{F}^L$, outputs a proof matrix $\Pi \in \mathbb{F}^{m \times d}$, where d and m depend only on C.
- Query$(\mathbb{F}, C)$ is a PPT algorithm that given an arithmetic verification circuit C outputs a query vector $\vec{q} \in \mathbb{F}^m$.
- Verify$(\mathbb{F}, x, \vec{q}, \vec{a})$ is a polynomial-time algorithm that given input $x \in \mathbb{F}^\ell$, query vector $\vec{q}$, and answer vector $\vec{a}$, outputs acc or rej.

The algorithms (Prove, Query, Verify) should satisfy the following:

- **Completness.** For any arithmetic circuit $C : \mathbb{F}^\ell \times \mathbb{F}^L \to \mathbb{F}$, input $x \in \mathbb{F}^\ell$ and witness $w \in \mathbb{F}^L$ such that $C(x, w) = 0$ we have $\Pr[\Pi \overset{R}{\leftarrow} \text{Prove}(\mathbb{F}, C, x, w), \vec{q} \overset{R}{\leftarrow} \text{Query}(\mathbb{F}, C) : \text{Verify}(\mathbb{F}, x, \vec{q}, \vec{q} \cdot \Pi) = \text{acc}] = 1$.
- **Reusable ϵ-soundness.** For any $C : \mathbb{F}^\ell \times \mathbb{F}^L \to \mathbb{F}$, input $x \in \mathbb{F}^\ell$ such that $C(x, w) \neq 0$ for all $w \in \mathbb{F}^L$, adversarially chosen $\Pi^* \in \mathbb{F}^{m \times d}$ and vector $\vec{b}^* \in \mathbb{F}^d$, we have $\Pr[\vec{q} \overset{R}{\leftarrow} \text{Query}(\mathbb{F}, C) : \text{Verify}(\mathbb{F}, x, \vec{q}, \vec{q} \cdot \Pi^* + \vec{b}^*) = \text{acc}] \leq \epsilon$. Moreover, for every $\mathbb{F}, C, x, \Pi^*, \vec{b}^*$ the probability of Verify accepting (over the choice of $\vec{q}$) is either 1 or $\leq \epsilon$. Unless otherwise specified, we assume that $\epsilon \leq O(|C|/|\mathbb{F}|)$.
- **Honest-verifier zero-knowledge.** There exists a PPT simulator Sim such that for any arithmetic circuit $C : \mathbb{F}^\ell \times \mathbb{F}^L \to \mathbb{F}$, input $x \in \mathbb{F}^\ell$, and witness $w \in \mathbb{F}^L$ such that $C(x, w) = 0$, the output of $\text{Sim}((\mathbb{F}, C, x)$ is a pair $(\vec{q}, \vec{a})$ that is identically distributed to $\{ (\vec{q}, \vec{a}) : \Pi \overset{R}{\leftarrow} \text{Prove}(\mathbb{F}, C, x, w), \vec{q} \overset{R}{\leftarrow} \text{Query}(\mathbb{F}, C); \vec{a} \leftarrow \vec{q} \cdot \Pi \}$.

Note that the final requirement in the definition of reusable soundness guarantees that even by observing the verifier's behavior on a maliciously chosen input x^* and proof Π^*, the prover cannot obtain significant information about the query $\vec{q}$. This ensures that $\vec{q}$ can be reused without compromising soundness. We note that our proofs also satisfy the *knowledge* property as defined in [14]. We focus here on soundness for simplicity.

From HVZK-LIP to reusable NIZK over VOLE. We now describe a simple transformation from any HVZK-LIP to reusable NIZK in the VOLE-hybrid model, where the prover plays the role of the VOLE sender P_0 and the verifier plays the role of the VOLE receiver P_1. The verifier's VOLE inputs x_i depend only on the query $\vec{q}$. This allows us to reuse the same x_i for multiple proofs, where each proof instance j uses fresh values of $(\vec{u}_i^j, \vec{v}_i^j)$ to mask the proof matrix Π.

The main idea behind the transformation is that the matrix-vector product $\vec{a} = \vec{q} \cdot \Pi$ can be encoded by $\vec{a}_i = (q_i \cdot \Pi_i + \vec{b}_i)$, $1 \leq i \leq m$, together with $\vec{c} = \sum \vec{b}_i$, where Π_i is the i-th column of Π and the $\vec{b}_i$ are random vectors in $\mathbb{F}^d$. Indeed, it is easy to check that $\vec{a} = \sum_{i=1}^m \vec{a}_i - \vec{c}$, and the information available to the verifier (namely, $\vec{q}, \vec{a}_i, \vec{c}$) reveals no information about Π other than $\vec{a}$. Thus,

[4] Since our NIZK protocols are *proof* systems for NP (rather than arguments), there is no hope to make them succinct [43]. Moreover, the assumptions on which we rely (LPN and OT) are not known to imply even collision-resistant hash functions, let alone succinct arguments for NP.

the value of $\vec{a}$ can be transferred to the prover via m instances of VOLE of length d, where the VOLE inputs of the prover (sender) are $(\Pi_i, \vec{b}_i)$ and the VOLE inputs of the verifier (receiver P_1) are q_i. Completeness and honest-verifier zero-knowledge are directly inherited from the HVZK-LIP via the properties of the encoding discussed above. Soundness follows by observing that any maliciously chosen $(\vec{u}_i^*, \vec{v}_i^*)$ that the prover feeds as inputs to the VOLE instances in the NIZK protocol and any message $\vec{c}^*$ have the same effect as using the matrix Π^* such that $\Pi_i^* = \vec{u}_i^*$ and the offset $\vec{b}^* = \sum \vec{v}_i^* - \vec{c}^*$ in the HVZK-LIP protocol. This construction of NIZK from a VOLE generator is formally described in Figure 3.

NIZK protocol from VOLE generator

- **Building blocks:** VOLE generator (Setup, Expand); HVZK-LIP (Prove, Query, Verify) with answer length d.
- **Setup:** Given a verification circuit C over $\mathbb{F}$ and a bound T on the number of statements, securely generate the following correlated randomness:
  - Let $(q_1, \ldots, q_m) \xleftarrow{R} \mathsf{Query}(\mathbb{F}, C)$.
  - For $i = 1, \ldots, m$ and $n = dt$, let $(\mathsf{seed}_0^i, \mathsf{seed}_1^i) \xleftarrow{R} \mathsf{Setup}(1^\lambda, \mathbb{F}, n = dT, q_i)$.
  - For $i = 1, \ldots, m$, Prover gets seed_0^i, Verifier gets seed_1^i and q_i.
- **Local computation:** For $i = 1, \ldots, m$, Prover computes $(\vec{u}_i, \vec{v}_i) \xleftarrow{R} \mathsf{Expand}(0, \mathsf{seed}_0^i)$ and Verifier computes $\vec{w}_i = \mathsf{Expand}(1, \mathsf{seed}_1^i)$. Parse each $\vec{u}_i$ as $(\vec{u}_i^j)$, $1 \le j \le T$, where $\vec{u}_i^j \in \mathbb{F}^d$, and similarly for $\vec{v}_i, \vec{w}_i$.
- **Prover algorithm:** For $1 \le j \le T$, on input (x^j, w^j), Prover lets $\Pi^j \xleftarrow{R} \mathsf{Prove}(\mathbb{F}, C, x^j, w^j)$. It sends a proof $\pi^j = (\vec{a}_1^j, \ldots, \vec{a}_m^j, \vec{c} = \sum_{i=1}^m \vec{v}_i^j)$, where $\vec{a}_i^j = \Pi_i^j - \vec{u}_i^j$.
- **Verifier algorithm:** For $1 \le j \le T$, on input x^j, Verifier lets $\vec{a}^j = \sum_{i=1}^m (q_i \cdot \vec{a}_i^j + \vec{w}_i^j) - \vec{c}$. It decides whether to accept by running $\mathsf{Verify}(\mathbb{F}, x^j, (q_1, \ldots, q_m), \vec{a}^j)$.

Figure 3: NIZK with reusable setup from VOLE generator.

Instantiations. As shown in [14], any linear PCP can be compiled into an HVZK-LIP with a very small overhead. In particular, the QAP-based linear PCP of GGPR [35] implies an HVZK-LIP proving the satisfiability of arithmetic circuit C of size s over $\mathbb{F}$ with parameters $m = O(s)$, $d = 4$, and $\epsilon = O(s/|\mathbb{F}|)$, where the proof Π is generated from (x, w) in time quasi-linear in s. This results in NIZK protocols in which $O(s)$ instances of a VOLE generator can be used to non-interactively prove any polynomial number of statements $C(x^j, \cdot)$, and where each proof contains $O(s)$ field elements. One can further improve the prover's time complexity from quasi-linear to linear by partitioning the circuit gates into constant-size blocks and applying an instance of the GGPR-based LPCP (or even the simpler "Hadamard-based LPCP" [47]) separately to each block. This optimization exploits the fact that we give up on succinctness in our setting. We leave the refined tuning of parameters and implementation of our NIZK technique to future work.

Comparison with designated verifier NIZK.. It is instructive to compare our NIZK protocols to designated-verifier NIZK protocols from the literature: see [20] and references therein. Our NIZK protocol is weaker in that it relies on a stronger setup: whereas in standard designated verifier NIZK a verifier can post a public key that can be used by many different provers, our setup requires correlated randomness or interaction between a designated verifier and a designated prover. However, in cases where the same prover proves many statements to the same verifier, the amortized cost of this setup is small. The main advantage of our protocol is that its online phase is very lightweight and does not involve public key cryptography. In fact, if the Expand function of the VOLE generator is invoked in the offline phase (without any interaction), computing and verifying each proof is less efficient than evaluating $C(x, w)$ in the clear by only a small constant factor. Our protocols are the first (reusable) NIZK protocols of any kind to rely on LPN, or alternatively LPN and OT if the setup is generated by a distributed protocol.

ACKNOWLEDGMENTS

Work supported by ERC grant 742754 (project NTSC). E. Boyle additionally supported by ISF grant 1861/16 and AFOSR Award FA9550-17-1-0069. G. Couteau additionally supported by ERC grant 724307 (project PREP-CRYPTO). N. Gilboa additionally supported by ISF grant 1638/15, and a grant by the BGU Cyber Center. Y. Ishai additionally supported by ISF grant 1709/14, NSF-BSF grant 2015782, and a grant from the Ministry of Science and Technology, Israel and Department of Science and Technology, Government of India.

REFERENCES

[1] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 430–454.

[2] Michael Alekhnovich. 2003. More on Average Case vs Approximation Complexity. In *44th FOCS*. IEEE Computer Society Press, 298–307.

[3] Benny Applebaum. 2012. Pseudorandom generators with long stretch and low locality from random local one-way functions. In *44th ACM STOC*, Howard J. Karloff and Toniann Pitassi (Eds.). ACM Press, 805–816.

[4] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. 2017. Secure Arithmetic Computation with Constant Computational Overhead *(LNCS)*. Springer, Heidelberg, 223–254.

[5] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. 2011. How to Garble Arithmetic Circuits. In *52nd FOCS*, Rafail Ostrovsky (Ed.). IEEE Computer Society Press, 120–129.

[6] Benny Applebaum and Shachar Lovett. 2016. Algebraic attacks against random local functions and their countermeasures. In *48th ACM STOC*, Daniel Wichs and Yishay Mansour (Eds.). ACM Press, 1087–1100.

[7] Sanjeev Arora and Rong Ge. 2010. Learning Parities with Structured Noise.. In *Electronic Colloquium on Computational Complexity (ECCC)*, Vol. 17. 66.

[8] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. 2003. A Fast Provably Secure Cryptographic Hash Function. Cryptology ePrint Archive, Report 2003/230. http://eprint.iacr.org/2003/230.

[9] Donald Beaver. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO'91 (LNCS)*, Joan Feigenbaum (Ed.), Vol. 576. Springer, Heidelberg, 420–432.

[10] Donald Beaver. 1995. Precomputing Oblivious Transfer. In *CRYPTO'95 (LNCS)*, Don Coppersmith (Ed.), Vol. 963. Springer, Heidelberg, 97–109.

[11] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. 2012. Decoding Random Binary Linear Codes in $2^{n/20}$: How 1 + 1 = 0 Improves Information Set Decoding. In *EUROCRYPT 2012 (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, Heidelberg, 520–536.

[12] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT 2011 (LNCS)*, Kenneth G. Paterson (Ed.), Vol. 6632. Springer, Heidelberg, 169–188.

[13] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. 2011. Smaller Decoding Exponents: Ball-Collision Decoding. In *CRYPTO 2011 (LNCS)*, Phillip Rogaway (Ed.), Vol. 6841. Springer, Heidelberg, 743–760.

[14] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. 2013. Succinct Non-interactive Arguments via Linear Interactive Proofs. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings.* 315–333. https://doi.org/10.1007/978-3-642-36594-2_18

[15] Avrim Blum, Adam Kalai, and Hal Wasserman. 2000. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*. ACM Press, 435–440.

[16] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. 2017. Homomorphic Secret Sharing: Optimizations and Applications. In *CCS 2017*. 2105–2122.

[17] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function Secret Sharing. In *EUROCRYPT 2015, Part II (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, Heidelberg, 337–367. https://doi.org/10.1007/978-3-662-46803-6_12

[18] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In *ACM CCS 16*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1292–1303.

[19] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2017. Group-Based Secure Computation: Optimizing Rounds, Communication, and Computation. In *Eurocrypt'17*. 163–193.

[20] Pyrros Chaidos and Geoffroy Couteau. 2018. Efficient Designated-Verifier Non-interactive Zero-Knowledge Proofs of Knowledge. In *EUROCRYPT 2018, Part III*. 193–221.

[21] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. 2018. Reusable Non-Interactive Secure Computation. Manuscript.

[22] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. 2018. SPD \mathbb Z_2^k : Efficient MPC mod 2^k for Dishonest Majority. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*. 769–798. https://doi.org/10.1007/978-3-319-96881-0_26

[23] Ronald Cramer, Ivan Damgård, and Yuval Ishai. 2005. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC 2005 (LNCS)*, Joe Kilian (Ed.), Vol. 3378. Springer, Heidelberg, 342–362.

[24] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. 2003. Efficient Multi-party Computation over Rings. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*. 596–613. https://doi.org/10.1007/3-540-39200-9_37

[25] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO 2012 (LNCS)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.), Vol. 7417. Springer, Heidelberg, 643–662.

[26] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. 2016. Spooky Encryption and Its Applications. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*. 93–122. https://doi.org/10.1007/978-3-662-53015-3_4

[27] Jack Doerner and Abhi Shelat. 2017. Scaling ORAM for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 523–535.

[28] Nico Döttling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto Trifiletti. 2017. TinyOLE: Efficient Actively Secure Two-Party Computation from Oblivious Linear Function Evaluation. In *ACM CCS 17*. ACM Press, 2263–2276.

[29] Erez Druk and Yuval Ishai. 2014. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *ITCS 2014*, Moni Naor (Ed.). ACM, 169–182.

[30] Niall Emmart, Justin Luitjens, Charles Weems, and Cliff Woolley. 2016. Optimizing modular multiplication for nvidia's maxwell gpus. In *Computer Arithmetic (ARITH), 2016 IEEE 23nd Symposium on*. IEEE, 47–54.

[31] Andre Esser, Robert Kübler, and Alexander May. 2017. LPN Decoded *(LNCS)*. Springer, Heidelberg, 486–514.

[32] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. 2017. Homomorphic Secret Sharing from Paillier Encryption. In *Provable Security - 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings*. 381–399. https://doi.org/10.1007/978-3-319-68637-0_23

[33] Matthieu Finiasz and Nicolas Sendrier. 2009. Security Bounds for the Design of Code-Based Cryptosystems. In *ASIACRYPT 2009 (LNCS)*, Mitsuru Matsui (Ed.), Vol. 5912. Springer, Heidelberg, 88–105.

[34] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. 303–324. https://doi.org/10.1007/978-3-540-30576-7_17

[35] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *EUROCRYPT*. 626–645.

[36] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. 2017. Maliciously Secure Oblivious Linear Function Evaluation with Constant Overhead. In *ASIACRYPT 2017, Part I (LNCS)*. Springer, Heidelberg, 629–659.

[37] Satrajit Ghosh and Tobias Nilges. 2017. An Algebraic Approach to Maliciously Secure Private Set Intersection. *IACR Cryptology ePrint Archive* 2017 (2017), 1064. http://eprint.iacr.org/2017/1064

[38] Niv Gilboa. 1999. Two Party RSA Key Generation. In *CRYPTO'99 (LNCS)*, Michael J. Wiener (Ed.), Vol. 1666. Springer, Heidelberg, 116–129.

[39] Niv Gilboa and Yuval Ishai. 1999. Compressing Cryptographic Resources. In *CRYPTO'99 (LNCS)*, Michael J. Wiener (Ed.), Vol. 1666. Springer, Heidelberg, 591–608.

[40] Niv Gilboa and Yuval Ishai. 2014. Distributed Point Functions and Their Applications. In *EUROCRYPT 2014 (LNCS)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.), Vol. 8441. Springer, Heidelberg, 640–658. https://doi.org/10.1007/978-3-642-55220-5_35

[41] Oded Goldreich. 2000. Candidate One-Way Functions Based on Expander Graphs. Cryptology ePrint Archive, Report 2000/063. http://eprint.iacr.org/2000/063.

[42] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229.

[43] Oded Goldreich, Salil Vadhan, and Avi Wigderson. 2002. On interactive proofs with a laconic prover. *Computational Complexity* 11, 1 (2002), 1–53. https://doi.org/10.1007/s00037-002-0169-0

[44] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. 2016. Secure Multiparty Computation with General Interaction Patterns. In *ITCS 2016*, Madhu Sudan (Ed.). ACM, 157–168.

[45] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. 2018. TinyKeys: A New Approach to Efficient Multi-Party Computation. In *CRYPTO*.

[46] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. 2005. Sufficient Conditions for Collision-Resistant Hashing. In *TCC*. 445–456. https://doi.org/10.1007/b106171

[47] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. 2007. Efficient Arguments without Short PCPs. In *CCC*. 278–291.

[48] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2004. Batch codes and their applications. In *36th ACM STOC*, László Babai (Ed.). ACM Press, 262–271.

[49] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2008. Cryptography with constant computational overhead. In *40th ACM STOC*, Richard E. Ladner and Cynthia Dwork (Eds.). ACM Press, 433–442.

[50] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2009. Zero-Knowledge Proofs from Secure Multiparty Computation. *SIAM J. Comput.* 39, 3 (2009), 1121–1152. https://doi.org/10.1137/080725398

[51] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2008. Founding Cryptography on Oblivious Transfer - Efficiently. In *CRYPTO 2008 (LNCS)*, David Wagner (Ed.), Vol. 5157. Springer, Heidelberg, 572–591.

[52] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2009. Secure Arithmetic Computation with No Honest Majority. In *TCC 2009 (LNCS)*, Omer Reingold (Ed.), Vol. 5444. Springer, Heidelberg, 294–314.

[53] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[54] Joe Kilian. 1988. Founding Cryptography on Oblivious Transfer. In *20th ACM STOC*. ACM Press, 20–31.

[55] Joe Kilian, Silvio Micali, and Rafail Ostrovsky. 1989. Minimum Resource Zero-Knowledge Proofs (Extended Abstract). In *FOCS '89*. 474–479.

[56] Kazuki Kobayashi and Tomoharu Shibuya. 2012. Generalization of Lu's linear time encoding algorithm for LDPC codes. In *Information Theory and its Applications (ISITA), 2012 International Symposium on*. IEEE, 16–20.

[57] Jin Lu and José MF Moura. 2010. Linear time encoding of LDPC codes. *IEEE Transactions on Information Theory* 56, 1 (2010), 233–249.

[58] Vadim Lyubashevsky. 2005. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Approximation, randomization and combinatorial optimization. Algorithms and techniques*. Springer, 378–389.

[59] Alexander May, Alexander Meurer, and Enrico Thomae. 2011. Decoding Random Linear Codes in $\tilde{O}(2^{0.054n})$. In *ASIACRYPT 2011 (LNCS)*, Dong Hoon Lee and Xiaoyun Wang (Eds.), Vol. 7073. Springer, Heidelberg, 107–124.

[60] Alexander May and Ilya Ozerov. 2015. On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes. In *EUROCRYPT 2015, Part I (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9056. Springer, Heidelberg,

203–228. https://doi.org/10.1007/978-3-662-46800-5_9

[61] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. 19–38. https://doi.org/10.1109/SP.2017.12

[62] Moni Naor and Benny Pinkas. 2006. Oblivious Polynomial Evaluation. *SIAM J. Comput.* 35, 5 (2006), 1254–1281.

[63] Eugene Prange. 1962. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory* 8, 5 (1962), 5–9.

[64] Peter Rindal and Mike Rosulek. 2016. Faster Malicious 2-Party Secure Computation with Online/Offline Dual Execution.. In *USENIX Security Symposium*. 297–314.

[65] Peter Scholl. 2018. Extending Oblivious Transfer with Low Communication via Key-Homomorphic PRFs *(LNCS)*. Springer, Heidelberg, 554–583.

[66] Daniel A Spielman. 1996. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory* 42, 6 (1996), 1723–1731.

[67] Jacques Stern. 1988. A method for finding codewords of small weight. In *International Colloquium on Coding Theory and Applications*. Springer, 106–113.

[68] Doug Stinson, Ruizhong Wei, and Maura Paterson. 2009. Combinatorial batch codes. *Advances in Mathematics of Communications* 3, 1 (2009), 13–27.

[69] Rodolfo Canto Torres and Nicolas Sendrier. 2016. Analysis of information set decoding for a sub-linear error weight. In *International Workshop on Post-Quantum Cryptography*. Springer, 144–161.

[70] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated garbling and efficient maliciously secure two-party computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 21–37.