Secure Arithmetic Computation with Constant Computational Overhead*

Benny Applebaum[1], Ivan Damgård[2], Yuval Ishai[3], Michael Nielsen[2], and Lior Zichron[1]

[1] Tel Aviv University, {bennyap@post,liorzichron@mail}.tau.ac.il
[2] Aarhus University, {ivan,mik}@cs.au.dk
[3] Technion and UCLA, yuvali@cs.technion.ac.il

Abstract. We study the complexity of securely evaluating an arithmetic circuit over a finite field $\mathbb{F}$ in the setting of secure two-party computation with semi-honest adversaries. In all existing protocols, the number of arithmetic operations per multiplication gate grows either linearly with $\log |\mathbb{F}|$ or poly-logarithmically with the security parameter. We present the first protocol that only makes a *constant* (amortized) number of field operations per gate. The protocol uses the underlying field $\mathbb{F}$ as a black box, and its security is based on arithmetic analogues of well-studied cryptographic assumptions.

Our protocol is particularly appealing in the special case of securely evaluating a "vector-OLE" function of the form $\boldsymbol{a}x+\boldsymbol{b}$, where $x \in \mathbb{F}$ is the input of one party and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^w$ are the inputs of the other party. In this case, which is motivated by natural applications, our protocol can achieve an asymptotic rate of 1/3 (i.e., the communication is dominated by sending roughly $3w$ elements of $\mathbb{F}$). Our implementation of this protocol suggests that it outperforms competing approaches even for relatively small fields $\mathbb{F}$ and over fast networks.

Our technical approach employs two new ingredients that may be of independent interest. First, we present a general way to combine any linear code that has a fast encoder and a cryptographic ("LPN-style") pseudorandomness property with another linear code that supports fast encoding *and erasure-decoding*, obtaining a code that inherits both the pseudorandomness feature of the former code and the efficiency features of the latter code. Second, we employ local *arithmetic* pseudo-random generators, proposing arithmetic generalizations of boolean candidates that resist all known attacks.

1 Introduction

There are many situations in which computations are performed on sensitive *numerical* data. A computation on numbers can usually be expressed as a sequence of arithmetic operations such as addition, subtraction, and multiplication.[4]

In cases where the sensitive data is distributed among multiple parties, this calls for *secure arithmetic computation*, namely secure computation of functions defined by arithmetic operations. It is convenient to represent such a function by an *arithmetic circuit*, which is similar to a standard boolean circuit except that gates are labeled by addition, subtraction, or multiplication. It is typically sufficient to consider such circuits that evaluate the operations over a large *finite field* $\mathbb{F}$, since arithmetic computations over the integers or (bounded precision) reals can be reduced to this case. Computing over finite fields (as opposed to integers or reals) can also be a feature, as it is useful for applications in threshold cryptography (see, e.g., [16, 27]). In the present work we are mainly interested in the case of secure arithmetic *two-party* computation in the presence of *semi-honest* adversaries.[5] From here on, the term "secure computation" will refer specifically to this case.

* An extended abstract of this paper appears in the Proceedings of Crypto 2017.

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

[5] Our results extend naturally to the case of secure multi-party computation with no honest majority. We restrict the attention to the two-party case for simplicity.

Oblivious Linear-function Evaluation. A natural complete primitive for secure arithmetic computation is Oblivious Linear-function Evaluation (OLE). OLE is a two-party functionality that receives a field element $x \in \mathbb{F}$ from Alice and field elements $a, b \in \mathbb{F}$ from Bob and delivers $ax + b$ to Alice. OLE can be viewed as the arithmetic analogue of 1-out-of-2 Oblivious Transfer of bits (bit-OT) [23]. In the binary case, every boolean circuit C can be securely evaluated with perfect security by using $O(|C|)$ invocations of an ideal bit-OT oracle via the "GMW protocol" [31, 28]. A simple generalization of this protocol can be used to evaluate any arithmetic circuit C over $\mathbb{F}$ using $O(|C|)$ invocations of OLE and $O(|C|)$ field operations [36].

The complexity of secure arithmetic computation. The goal of this work is to minimize the complexity of secure arithmetic computation. In light of the above, this reduces to efficiently realizing multiple instances of OLE. We start by surveying known approaches. The most obvious is a straightforward reduction to standard secure computation methods by emulating field operations using bit operations. This approach is quite expensive both asymptotically and in terms of concrete efficiency. In particular, it typically requires many "cryptographic" operations for securely emulating each field operation.

A more direct approach is via *homomorphic encryption*. Since OLE is a degree-1 function, it can be directly realized by using "linear-homomorphic" encryption schemes (that support addition and scalar multiplication). This approach can be instantiated using Paillier encryption [49, 19, 27] or using encryption schemes based on (ring)-LWE [44, 20]. While these techniques can be optimized to achieve good communication complexity, their concrete computational cost is quite high. In asymptotic terms, the best instantiations of this approach have *computational overhead* that grows polylogarithmically with the security parameter k. That is, the computational complexity of such secure computation protocols (in any standard computational model) is bigger than the computational complexity of the insecure computation by at least a polylogarithmic factor in k.

Another approach, first suggested by Gilboa [27] and recently implemented by Keller et al. [38], is to use a simple information-theoretic reduction of OLE to string-OT. By using a bit-decomposition of Alice's input x, an OLE over a field $\mathbb{F}$ with ℓ-bit elements can be perfectly reduced to ℓ instances of OT, where in each OT one of two field elements is being transferred from Bob to Alice. Using fast methods for OT extension [32, 13], the OTs can be implemented quite efficiently. However, even when neglecting the cost of OTs, the communication involves 2ℓ field elements and the computation involves $O(\ell)$ field operations per OLE. This overhead can be quite large for big fields $\mathbb{F}$ that are often useful in applications.

A final approach, which is the most relevant to our work, uses a computationally secure reduction from OLE to string-OT that assumes the peudorandomness of noisy random codewords in a linear code. This approach was first suggested by Naor and Pinkas [47] and was further developed by Ishai, Prabhakaran, and Sahai [36]. The most efficient instantiation of these protocols relies on the assumption that a noisy random codeword of a Reed-Solomon code is pseudorandom, provided that the noise level is sufficiently high to defeat known list-decoding algorithms. In the best case scenario, this approach has polylogarithmic computational overhead (using asymptotically fast FFT-based algorithms for encoding and decoding Reed-Solomon codes). See Section 1.3 for a more detailed overview of existing approaches and [36] for further discussion of secure arithmetic computation and its applications.

The above state of affairs leaves the following question open:

Is it possible to realize secure arithmetic computation with constant computational overhead*?*

To be a bit more precise, by "constant computational overhead" we mean that there is a protocol which can securely evaluate any arithmetic circuit C over any finite field $\mathbb{F}$, with a computational cost (on a RAM machine) that is only a constant times bigger than the cost of performing $|C|$ field operations with no security at all. Here we make the standard convention of viewing the size of C also as a security parameter, namely the view of any adversary running in time $\text{poly}(|C|)$ can be simulated up to a negligible error (in $|C|$). In the boolean case, Ishai, Kushilevitz, Ostrovsky, and Sahai [35] showed that secure computation with constant computational overhead can be based on the conjectured existence of a local polynomial-stretch pseudorandom generator (PRG). In contrast, in all known protocols for the arithmetic case the computational overhead either grows linearly with $\log |\mathbb{F}|$ or polylogarithmically with the security parameter.

1.1 Our Contribution

We improve both the asymptotic and the concrete efficiency of secure arithmetic computation. On the asymptotic efficiency front, we settle the above open question in the affirmative under plausible cryptographic assumptions. More concretely, our main result is a protocol that securely evaluates any arithmetic circuit C over $\mathbb{F}$ using only $O(|C|)$ field operations, independently of the size of $\mathbb{F}$. The protocol uses the underlying field $\mathbb{F}$ as a black box, where the number of field operations depends only on the security parameter and not on the field size.[6] The security of the protocol is based on arithmetic analogues of well-studied cryptographic assumptions: concretely, an arithmetic version of an assumption due to Alekhnovich [1] (or similar kinds of "LPN-style" assumptions) and an arithmetic version of a local polynomial-stretch PRG [35, 4, 11].[7]

On the concrete efficiency front, our approach is particularly appealing for a useful subclass of arithmetic computations that efficiently reduce to a multi-output extension of OLE that we call *vector-OLE*. A vector-OLE of width w is a two-party functionality that receives a field element $x \in \mathbb{F}$ from Alice and a pair of *vectors* $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^w$ from Bob and delivers $\boldsymbol{a}x + \boldsymbol{b}$ to Alice. We obtain a secure protocol for vector-OLE with constant computational overhead and with an asymptotic communication rate of $1/3$ (i.e., the ratio between w and the number of communicated field elements tends to $1/3$ as w tends to infinity). Our implementation of this protocol suggests that it outperforms competing approaches even for relatively small fields $\mathbb{F}$ and values of w, and even over fast networks. The protocol is also based on more conservative assumptions, namely it can be based only on the first of the two assumptions on which our more general result is based. This assumption is arguably more conservative than the assumption on noisy Reed-Solomon codes used in [47, 36], since the underlying codes do not have an algebraic structure that gives rise to efficient (list-)decoding algorithms.

Vector-OLE can be viewed as an arithmetic analogue of string-OT. Similarly to the usefulness of string-OT for garbling schemes [55], vector-OLE is useful for *arithmetic garbling* [10, 5] (see Section 4). Moreover, there are several natural secure computation tasks that can be directly and efficiently realized using vector-OLE. One class of such tasks are in the domain of secure linear algebra [18]. As a simple example, the secure multiplication of an $n \times n$ matrix by a length-n vector easily reduces to n instances of vector-OLE of width n. More generally, vector-OLE can be used in the OLE-based "arithmetic GMW" protocol [36] to efficiently handle the case of arithmetic circuits with large fan-out, where one field element is multiplied by w other field elements. Finally, a more specialized class of applications is in the domain of securely searching for nearest neighbors, e.g., in the context of secure face recognition [22]. The goal is to find in a database of n vectors of dimension d the vector which is closest in Euclidean distance to a given target vector. This task admits a simple reduction to d instances of width-n vector OLE, followed by non-arithmetic secure computation of a simple function (minimum) of n integers whose size is independent of d. The cost of such a protocol is dominated by the cost of vector-OLE. See Section 5 for a more detailed discussion of some of these applications.

1.2 Overview of Techniques

Our constant-overhead protocol for a general circuit C is obtained in three steps. The first step is a reduction of the secure computation of C to $n = O(|C|)$ instances of OLE via an arithmetic version of the GMW protocol.

The second step is a reduction of n instances of OLE to roughly $\sqrt{n}$ instances of vector-OLE of width $w = O(\sqrt{n})$. This step mimics the approach for constant-overhead secure computation of boolean circuits taken in [35], which combines a local polynomial-stretch PRGs with an information-theoretic garbling scheme [55, 33]. To extend this approach from the boolean to the arithmetic case, two changes are made. First, the information-theoretic garbling scheme for NC^0 is replaced by an arithmetic analogue [10]. More interestingly,

[6] The protocol additionally uses standard "bit-operations," but their complexity is dominated by the field operations for every field size.

[7] More precisely, we need a polynomial-stretch PRG with constant locality and *constant degree*, or equivalently, a polynomial-stretch PRG which can be computed by a constant-depth ($\mathbf{NC^0}$) arithmetic circuit. For brevity, through the introduction we refer to such a PRG as being *local* and implicitly assume the additional constant-degree requirement.

the polynomial-stretch PRGs in NC^0 needs to be replaced by an arithmetic analogue. We propose candidates for such arithmetic PRGs that generalize the boolean candidates from [29, 11] and can be shown to resist known classes of attacks. While the security of these PRG candidates remains to be further studied, there are no apparent weaknesses that arise from increasing the field size.

The final, and most interesting, step is a constant-overhead protocol for vector-OLE. As noted above, the protocol obtained in this step is independently useful for applications, and our implementation of this protocol beats competing approaches not only asymptotically but also in terms of its concrete efficiency.

Our starting point is the code-based OLE protocol from [47, 36]. This protocol can be based on any randomized linear encoding scheme E over $\mathbb{F}$ that has a the following "LPN-style" pseudorandomness property: If we encode a message $x \in \mathbb{F}$ and replace a small random subset of the symbols by uniformly random field elements, the resulting noisy codeword is pseudorandom. For most linear encoding schemes this appears to be a conservative assumption, since there are very few linear codes for which efficient decoding techniques are known. The OLE protocol proceeds by having Alice compute a random encoding $\boldsymbol{y} = E(x)$ and send a noisy version $\boldsymbol{y'}$ of $\boldsymbol{y}$ to Bob. Bob returns $\boldsymbol{c'} = a\boldsymbol{y'} + \boldsymbol{b}$ to Alice, where $\boldsymbol{b} = E(b)$ is a random linear encoding of b. Knowing the noise locations, Alice can decode $c = ax + b$ from $\boldsymbol{c'}$ via erasure-decoding in the linear code defined by E. If we ignore the noise coordinates, $\boldsymbol{c'}$ does not reveal to Alice any additional information about (a, b) except the output $ax + b$. However, the noise coordinates can reveal more information. To prevent this information from being leaked, Alice uses oblivious transfer (OT) to selectively learn only the non-noisy coordinates of $\boldsymbol{c'}$.

An attempt to extend the above protocol to the case of vector-OLE immediately encounters a syntactic difficulty. If the single value a is replaced by a vector $\boldsymbol{a}$, it is not clear how to "multiply" $\boldsymbol{y'}$ by $\boldsymbol{a}$. A workaround taken in [36] is to use a "multiplicative" encoding scheme E based on Reed-Solomon codes. The encoding and decoding of these codes incurs a polylogarithmic computational overhead, and the high noise level required for defeating known list-decoding algorithms results in a poor concrete communication rate. The algebraic nature of the codes also makes the underlying intractability assumption look quite strong. It is therefore desirable to base a similar protocol on other types of linear codes.

Our first idea for using general linear codes is to apply "vector-OLE reversal." Concretely, we apply a simple protocol for reducing vector-OLE to the computation of $\boldsymbol{a}x + \boldsymbol{b}$ where $\boldsymbol{a}$ is the input of Bob, x and $\boldsymbol{b}$ the are the inputs of Alice, and the output is delivered to Bob. Now a general linear encoding E can be used by Bob to encode its input $\boldsymbol{a}$, and since x is a scalar Alice can multiply the encoding by x and add an encoding of $\boldsymbol{b}$. If we base E on a linear-time encodable and decodable code, such as the code of Spielman [52], the protocol can be implemented using only $O(w)$ field operations. The problem with this approach the is that the pseudorandomness assumption looks questionable in light of the existence of an efficient decoding algorithm for E. Even if the noise can chosen in a way that still respects linear-time erasure-decoding but makes error-correction intractable (which is not at all clear), this would require a high noise rate and hurt the concrete efficiency.

Our final idea, which may be of independent interest, is that instead of requiring a single encoding E to simultaneously satisfy both "hardness" and "easiness" properties, we can combine two types of encoding to enjoy the best of both worlds. Concretely, we present a general way to combine any linear code C_1 that has a fast encoder and a cryptographic ("LPN-style") pseudorandomness property with another linear code C_2 that supports fast encoding and erasure-decoding (but has no useful hardness properties) to get a randomized linear encoding E that inherits the pseudorandomness feature from C_1 and the efficiency features from C_2. This is achieved by using a noisy output of C_1 to mask the output of C_2, which we pad with a sufficient number of 0s. Given the knowledge of the noise locations in the padding zone, the entire C_1 component can be recovered in a "brute-force" way via Gaussian elimination, and one can then compute and decode the output of C_2. When the expansion of E is sufficiently large, the Gaussian elimination is only applied to a short part of the encoding length and hence does not form an efficiency bottleneck. Using these ideas, we obtain a constant-overhead vector-OLE protocol under a seemingly conservative assumption, namely a natural arithmetic analogue of an assumption due to Alekhnovich [1] or a similar assumption for other linear-time encodable codes, such as the ones proposed in [21]. The assumption is conservative in that it can apply to codes that do not have the special structure required for fast erasure-decoding.

1.3 Related Work

In this section we give an overview of known techniques for OLE (with semi-honest security) and compare these previous techniques to our approach.

First, the work of Gilboa [27] (see also [38]) implements OLE in a field with n-bit elements using n oblivious transfers of field elements. The asymptotic communication complexity of this approach is larger than ours by a factor $\Omega(n)$.

To be more concrete, in the case of implementing vector-OLE, we can get the following efficiency. Our vector-OLE implementation uses wn/r bits of communication to implement a width-w vector-OLE on n-bit field elements, where the rate r can be asymptotically as high as $1/3$ and is between $1/5$ and $1/10$ for our implementation.[8] The OT based approach will need to send at least wn^2 bits to do the same. So in cases where network bandwidth is the bottleneck, we can expect to be faster than the OT based approach by a factor of nr. This might happen in practice, as can be seen from our experiments. They indicate that on the platform we used, for $32 \leq n \leq 1024$ and aiming for 80 or 100 bits of security, our implementation uses around 20-50 Mbits/sec of bandwidth. Hence if network speed was lower than this, then on the same machines the OT based approach would be slower than ours as we just explained.

Actually, our vector-OLE implementation outperforms the OT based approach even if communication is not the bottleneck: The latest timings for semi-honest string OT on the type of architecture we used (2 desktop computers connected by a LAN) are from [13] (see also [38]) and indicate that one OT can be done in amortized time of about 0.2 μs, so that $0.2n\mu s$ would an estimate for the time needed for one OLE. In contrast, our running times (for 100-bit security) are much faster, even for the smallest case we considered ($n = 32$) we have 0.7 μs amortized time per OLE. For larger fields, the picture is similar, for instance for $n = 1024$, we obtain $19.5\mu s$ per OLE, where the estimate for the OT based technique is about $200\mu s$. This comparison is accurate for vector-OLE of width $w = 20.000$ as we used in our experiments for 100 bit security. We can trivially get smaller w by simply not using some of the w OLEs we generate. Doing the math we find that we can go to $w > 2000$ and still be faster than the OT-approach.

A second class of OLE protocols can be obtained from homomorphic encryption schemes: one party encrypts his input under his own key and sends the ciphertext to the other party. He can now multiply his input "into the ciphertext" and randomize it appropriately before returning it for decryption. This will work based on Paillier encryption (see, e.g., [22] for an application of this) but will be very slow because exponentiation is required for the plaintext multiplication. A more efficient approach is to use (ring)-LWE based schemes, as worked out in [20] by Damgård et al. Here the asymptotic communication overhead is worse than ours by a poly-logaritmic factor, at least for prime fields if one uses the so-called SIMD variant where the plaintext is a vector of field elements. However, the approach becomes very problematic for extension fields of large degree because key generation requires that we find a cyclotomic polynomial that splits in a very specific way modulo the characteristic, and one needs to go to very large keys before this happens. Quantifying this precisely is non-trivial and was not done in [20], but as an example, the overhead in ciphertext size is a factor of about 7 for a 64-bit prime fields, but is 1850 for $\mathbb{F}_{2^8}$. Also, the computational overhead for ring-LWE based schemes is much higher than ours: even if we pack as many field elements, say λ, into a ciphertext as possible, the overhead involved in encryption and decryption is superlinear in λ. Further λ needs to grow with the field size, again the asymptotic growth is hard to quantify exactly, but it is definitely super logarithmic. In more concrete terms, the computational overhead of homomorphic encryption makes these protocols slower in practice than the pure OT-based approach (see [38]), which is in turn generally slower than our approach for the case of vector-OLE.

A final class of protocols is more closely related to ours, namely the code-based approach of Naor and Pinkas [47] and its generalizations from [36]. The most efficient instantiation of these protocols relies on the assumption that noisy Reed-Solomon codewords are pseudo-random, whereas we can rely on a similar assumption for arbitrary linear codes, including linear-time encodable codes that are generated by a sparse matrix. Because known algorithms for encoding and decoding Reed-Solomon codes require quasi-linear time,

[8] The reason these rates are worse than our asymptotic rate of $1/3$ is that when implementing, we chose to sacrifice optimal rate in return for better concrete computational efficiency. See Section 6 for details. Different implementation choices can make the rate closer to $1/3$.

these protocols (even when applied to the case of vector-OLE) are asymptotically slower than ours by a poly-logarithmic factor. As for the communication, we obtain an asymptotic rate of 1/3 for vector-OLE, whereas the rate of the protocol from [36] is a significantly smaller constant: one loses a factor 2 because the protocol involves point-wise multiplication of codewords, so codewords need to be long enough to allow decoding of a Reed-Solomon code based on polynomials of double degree. Even more significantly, on top of the above, the distance needs to be increased (so the rate decreases) to protect against attacks that rely on efficient list-decoding algorithms for Reed-Solomon codes. We can avoid this class of attacks by relying on codes that have no algebraic structure.

An advantage of the OLE protocols based on noisy Reed-Solomon codes from [47, 36] is that they yield constant-rate (non-vector) OLE directly, without overhead of converting vector-OLE to OLE via an arithmetic local PRG. Thus, we expect our technique to yield better *concrete* efficiency only in the case of vector-OLE.

2 Preliminaries

2.1 The arithmetic setting

Our formalization of secure arithmetic computation follows the one from [36], but simplifies it to account for the simpler setting of security against semi-honest adversaries. We also refine the computational model to allow for a more concrete complexity analysis. We refer the reader to [36] for more details.

Functionalities. We represent the functionalities that we want to realize securely via a multi-party variant of arithmetic circuits.

Definition 1 (Arithmetic circuits). *An* arithmetic circuit *C has the same syntax as a standard boolean circuit, except that the gates are labeled by '+' (addition), '-' (subtraction) or '*' (multiplication). Each input wire can be labeled by an input variable x_i or a constant $c \in \{0,1\}$. Given a finite field $\mathbb{F}$, an arithmetic circuit C with n inputs and m outputs naturally defines a function $C^{\mathbb{F}} : \mathbb{F}^n \to \mathbb{F}^m$. An arithmetic* functionality circuit *is an arithmetic circuit whose inputs and outputs are labeled by party identities. In the two-party case, such a circuit C naturally defines a two-party functionality $C^{\mathbb{F}} : \mathbb{F}^{n_1} \times \mathbb{F}^{n_2} \to \mathbb{F}^{m_1} \times \mathbb{F}^{m_2}$. We denote by $C^{\mathbb{F}}(x_1, x_2)_P$ the output of Party P on inputs (x_1, x_2).*

Protocols and complexity. To allow for a concrete complexity analysis, we view a protocol as a finite object that is generated by a protocol compiler (defined below). We assume that field elements have an adversarially chosen representation by ℓ-bit strings, where the protocol can depend on ℓ (but not on the representation). The representation is needed for realizing our protocols in the plain model. When considered as protocols in the OT-hybrid model, our protocols can be cast in the more restrictive arithmetic model of Applebaum et al. [5], where the parties do not have access to the bit-length of field elements or their representation, but can still perform field operations and communicate field elements over the OT channel. Protocols in this model have the feature that the number of field operations is independent of the field size.

By default, we model a protocol by a RAM program.[9] The choice of computational model does not change the number of field operations, which anyway dominates the overall cost as the field grows. In our theorem statements we will only refer to the number of field operations T, with the implicit understanding that all other computations can be implemented using $O(T\ell)$ bit operations. (Note that $T\ell$ bit operations are needed just for writing the outputs of T field operations.)

Protocol compiler. A *protocol compiler* $\mathcal{P}$ takes a security parameter 1^k, an arithmetic (two-party) functionality circuit C and bit-length parameter ℓ as inputs, and outputs a protocol Π that realizes C given an oracle to any field $\mathbb{F}$ whose elements are represented by ℓ-bit strings. It should satisfy the following correctness and security requirements.

[9] This choice is related to our use of the linear-time decoding algorithm of Spielman [52], which can only be implemented in linear time in the RAM model (and requires quasi-linear circuit size).

– Correctness: For every choice of $k, C, \mathbb{F}, \ell$, any representation of elements of $\mathbb{F}$ by ℓ-bit strings, and every possible pair of inputs (x_1, x_2) for C, the execution of Π on (x_1, x_2) ends with the parties outputting $C(x_1, x_2)$, except with negligible probability in k.
– Security: For every polynomial-size non-uniform $\mathcal{A}$ there is a negligible function ϵ such that the success probability of $\mathcal{A}$ in the following game is bounded by $1/2 + \epsilon(k)$:
  - On input 1^k, the adversary $\mathcal{A}$ picks a functionality circuit C, positive integer ℓ and field $\mathbb{F}$ whose elements are represented by ℓ-bit strings. The representation of field elements and field operations are implemented by a circuit $\mathcal{F}$ output by $\mathcal{A}$. (Note that all of the above parameters, including the complexity of the field operations, are effectively restricted to be polynomial in k.)
  - Let $\Pi^{\mathcal{F}}$ be the protocol returned by the compiler $\mathcal{P}$ on $1^k, C, \ell$, instantiating the field oracle $\mathbb{F}$ using $\mathcal{F}$.
  - $\mathcal{A}$ picks a corrupted party $P \in \{1, 2\}$ and two input pairs $x^0 = (x_1^0, x_2^0)$,$x^1 = (x_1^1, x_2^1)$ such that $C^{\mathbb{F}}(x^0)_P = C^{\mathbb{F}}(x^1)_P$.
  - Challenger picks a random bit b.
  - $\mathcal{A}$ is given the view of Party P in $\Pi^{\mathcal{F}}(x^b)$ and outputs a guess for b.

OLE and vector OLE. We will be particularly interested in the following two arithmetic computations: an OLE takes an input $x \in \mathbb{F}$ from Alice and a pair $a, b \in \mathbb{F}$ from Bob and delivers $ax + b$ to Alice. Vector OLE of width w is similar, except that the input of Bob is a pair of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^w$ and the output is $\boldsymbol{a}x + \boldsymbol{b}$. OLE and vector OLE can be viewed as arithmetic analogues of bit-OT and string-OT, respectively. Indeed, in the case $\mathbb{F} = \mathbb{F}_2$, the OLE functionalities coincide with the corresponding OT functionalities up to a local relabeling of the inputs. An arithmetic generalization of the standard "GMW Protocol" [31, 36] compiles any arithmetic circuit functionality C into a perfectly secure protocol that makes $O(s_\times)$ calls to an ideal OLE functionality, where $s_\times$ is the number of multiplication gates, and $O(|C|)$ field operations. Hence, to securely compute C with $O(|C|)$ field operations in the plain model it suffices to realize n instances of OLE using $O(n)$ field operations.

2.2 Decomposable affine randomized encoding (DARE)

Let $f : \mathbb{F}^\ell \to \mathbb{F}^m$ where $\mathbb{F}$ is some finite field.[10] We say that a function $\hat{f} : \mathbb{F}^\ell \times \mathbb{F}^\rho \to \mathbb{F}^m$ is a *perfect randomized encoding* [33, 8] of f if for every input $x \in \mathbb{F}^\ell$, the distribution $\hat{f}(x; r)$ induced by a uniform choice of $r \xleftarrow{\$} \mathbb{F}^\rho$, "encodes" the string $f(x)$ in the following sense:

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

We say that $\hat{f}(x; r)$ is decomposable and affine if $\hat{f}$ can be written as $\hat{f}(x; r) = (\hat{f}_0(r), \hat{f}_1(x_1; r), \ldots, \hat{f}_n(x_\ell; r))$ where $\hat{f}_i$ is linear in x_i, i.e., it can be written as $\boldsymbol{a}_i x_i + \boldsymbol{b}_i$ where the vectors $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ arbitrarily depend on the randomness r.

It follows from [34] (cf. [10]) that every single-output function $f : \mathbb{F}^d \to \mathbb{F}$ which can be computed by constant-depth circuit (aka $\mathbf{NC^0}$ function) admits a decomposable encoding which can be encoded and decoded by an arithmetic circuit of finite complexity D which depends only in the circuit depth. Note that any multi-output function can be encoded by concatenating independent randomized encodings of the functions defined by its output bits. Thus, we have the following:

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

Fact 1 *Let $f : \mathbb{F}^\ell \to \mathbb{F}^m$ be an $\mathbf{NC^0}$ function. Then, f has a DARE $\hat{f}$ which can be encoded, decoded and simulated by an arithmetic circuit of size $O(m)$ where the constant in the big-O notation depends on the circuit depth.*[11]

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

3 Vector OLE of large width

In this section, our goal is to construct a semi-honest secure protocol for Vector OLE of width w over the field $\mathbb{F} = \mathbb{F}_p$ for parties Alice and Bob.

As a stepping stone, we will first implement a "reversed" version of this that can easily be turned into what we actually want: for the Reverse vector OLE functionality, Bob has input $\boldsymbol{a} \in \mathbb{F}^w$, while Alice has input $x \in \mathbb{F}, \boldsymbol{b} \in \mathbb{F}^w$, and the functionality outputs nothing to Alice and $\boldsymbol{a} \cdot x + \boldsymbol{b}$ to Bob. The latter will be based on a special gadget (referred to as *fast hard/easy code*) that allows fast encoding and decoding under erasures but semantically hides the encoded messages in the presence of noise. We describe first this gadget and then give the protocol.

3.1 Ingredients

The main ingredient we need is a public matrix M over $\mathbb{F}$ with the following pseudorandomness property: If we take a random vector $\boldsymbol{y}$ in the image of M, and perturb it with "noise", the resulting vector $\hat{\boldsymbol{y}}$ is computationally indistinguishable from a truly random vector over $\mathbb{F}$. Our noise distribution corresponds to the p-ary symmetric channel with crossover over probability μ, that is, $\hat{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{e}$ where for each coordinate of $\boldsymbol{e}$ we assign independently the value zero with probability $1 - \mu$ and a uniformly chosen non-zero element from $\mathbb{F}$ with probability μ. We let $\mathcal{D}(\mathbb{F})^t_\mu$ denote the corresponding noise distributions for vectors of length t (and occasionally omit the parameters $\mathbb{F}, \mu$ and t when they are clear from the context). For concreteness, the reader may think of μ as a small constant, say $1/4$, however μ can also be chosen so that it tends to 0 when the security parameter k tends to infinity. The properties needed for our protocol are summarized in the following assumption, that will be discussed in Section 7.

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

1. *(Linear-time computation) The mapping $f_M : \boldsymbol{x} \mapsto M\boldsymbol{x}$ can be computed in linear-time in the output length. Formally, we assume that the sampler outputs a description of an $O(k^3)$-size arithmetic circuit over $\mathbb{F}$ for computing f_M.*
2. *(Pseudorandomness) The following ensembles (indexed by k) are computationally indistinguishable for* $\mathrm{poly}(k)$ *adversaries*
$$(M, M\boldsymbol{r} + \boldsymbol{e}) \qquad \text{and} \qquad (M, \boldsymbol{z}),$$
where $M \xleftarrow{\$} \mathcal{M}(1^k, p)$, $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}_p^k$, $\boldsymbol{e} \xleftarrow{\$} \mathcal{D}_\mu(\mathbb{F}_p)^\ell$ and $\boldsymbol{z} \xleftarrow{\$} \mathbb{F}_p^{k^3}$.
3. *(Linear independence) If we sample $M \xleftarrow{\$} \mathcal{M}(1^k, \mathbb{F})$ and keep each of the first $k \log^2 k$ rows independently with probability μ (and remove all other rows), then, except with negligible probability in k, the resulting matrix has full rank.*

We will also need a linear error correcting code Ecc over $\mathbb{F}$ with constant rate R and linear time encoding and decoding, where we only need decoding from a constant fraction of erasures μ' which is slightly larger than the noise rate μ. (For $\mu = 1/4$ we can take $\mu' = 1/3$.) Such codes are known to exist (cf. [52]) and can be efficiently constructed given a black-box access to $\mathbb{F}$.

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

Fast hard/easy code We combine the "fast code" Ecc and the "fast pseudorandom code" $\mathcal{M}$ into a single gadget that provides fast encoding and decoding under erasures, but hides the encoded message when delivered through a noisy channel. The gadget supports messages of length $w = \Theta(k^3)$. Our gadget is initialized by sampling a $k^3 \times k$ matrix M over $\mathbb{F}$ using the randomized algorithm $\mathcal{M}$ promised in Assumption 2. We view the matrix M as being composed of two matrices M^{top} with $u = 2k\log^2 k$ rows and k columns, placed above M^{bottom} which has $v = k^3 - u$ rows and k columns. Let $w = Rv = \Theta(k^3)$ be a message length parameter (corresponding to the width of the vector-OLE). Note that our Ecc encodes vectors of length w into vectors of length v.

For a message $\boldsymbol{a} \in \mathbb{F}^w$, and random vector $\boldsymbol{r} \in \mathbb{F}^k$, define the mapping

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

where $\circ$ denotes concatenation (and so $(0^u \circ \mathsf{Ecc}(\boldsymbol{a}))$ is a vector of length $u + v$). We will make use of the following useful properties of E:

1. (Fast and Linear) The mapping $E_{\boldsymbol{r}}(\boldsymbol{a})$ can be computed by making only $O(k^3)$ arithmetic operations. Moreover, it is a linear function of $(\boldsymbol{r}, \boldsymbol{a})$ and so $E_{\boldsymbol{r}}(\boldsymbol{a}) + E_{\boldsymbol{r'}}(\boldsymbol{a'}) = E_{\boldsymbol{r}+\boldsymbol{r'}}(\boldsymbol{a} + \boldsymbol{a'})$.
2. (Hiding under errors) For any message $\boldsymbol{a}$ and $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^k$ $\boldsymbol{e} \xleftarrow{\$} \mathcal{D}(\mathbb{F})_\mu^{k^3}$, the vector $E_{\boldsymbol{r}}(\boldsymbol{a}) + \boldsymbol{e}$ is pseudorandom and, in particular, it computationally hides $\boldsymbol{a}$.
3. (Fast decoding under erasures) Given a random $(1-\mu)$-subset I of the coordinates of $\boldsymbol{z} = E_{\boldsymbol{r}}(\boldsymbol{a})$ (i.e., each coordinate is erased with independently probability μ) it is possible to recover the vector $\boldsymbol{a}$, with negligible error probability, by making only $O(|\boldsymbol{z}|) = O(k^3)$ arithmetic operations. Indeed, letting I_0 (resp., I_1) denote the coordinates received from the u-prefix of $\boldsymbol{z}$ (resp., v-suffix of $\boldsymbol{z}$), we first recover $\boldsymbol{r}$ by solving the linear system $\boldsymbol{z}_{I_0} = (M^{\mathsf{top}}\boldsymbol{r})_{I_0}$ via Gaussian elimination in $O(k^3)$ arithmetic operations. By Assumption 2 (property 3) the system is likely to have a unique solution. Then we compute $(M^{\mathsf{bottom}}\boldsymbol{r})_{I_1}$ in time $O(k^3)$, subtract from $(E_{\boldsymbol{r}}(\boldsymbol{a}))_{I_1}$ to get $\mathsf{Ecc}(\boldsymbol{a})_{I_1}$, from which $\boldsymbol{a}$ can be recovered by erasure decoding in time $O(k^3)$.

3.2 From Fast hard/easy code to reverse vector-OLE

Our protocol uses the gadget E to implement a reversed vector-OLE. In the following we assume that the parties have access to a variant Oblivious Transfer of field elements which we assume (for now) is given as an ideal functionality. To be precise, the variant we need is one where Alice sends a field element f, Bob chooses to receive f, or to receive nothing, while Alice learns nothing new.

We describe the protocol under the assumption that the width w is taken to be $\Theta(k^3)$. A general value of w will be treated either by padding or by partitioning into smaller blocks of size $O(k^3)$ each. (See the proof of Theorem 3.)

Construction 1 (Reverse Vector OLE protocol) *To initialize the protocol one of the parties samples the matrix $M \xleftarrow{\$} \mathcal{M}(1^k, \mathbb{F})$ and publish it. The gadget E (and the parameters u, v and w) are now defined based on M and k as described above.*

1. *Bob has input $\boldsymbol{a} \in \mathbb{F}^w$. He selects random $\boldsymbol{r} \in \mathbb{F}^k$, chooses $\boldsymbol{e}$ according to $\mathcal{D}(\mathbb{F})_\mu^{u+v}$ and sends to Alice the vector $\boldsymbol{c} = E_{\boldsymbol{r}}(\boldsymbol{a}) + \boldsymbol{e}$.*
2. *Alice has input $x, \boldsymbol{b}$. She chooses $\boldsymbol{r'} \in \mathbb{F}^k$ at random and computes $\boldsymbol{d} = x \cdot \boldsymbol{c} + E_{\boldsymbol{r'}}(\boldsymbol{b})$.*
3. *Let I be an index set that contain those indices i for which $\boldsymbol{e}_i = 0$. These are called the noise free positions in the following. The parties now execute, for each entry i in $\boldsymbol{d}$, an OT where Alice sends $\boldsymbol{d}_i$. If $i \in I$, Bob chooses to receive $\boldsymbol{d}_i$, otherwise he chooses to receive nothing.*
4. *Notice that, since the function E is linear, we have*

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

Using subscript-I to denote restriction to the noise-free positions, what Bob has just received is

$$\boldsymbol{d}_I = (E_{\boldsymbol{s}}(x\boldsymbol{a} + \boldsymbol{b}))_I,$$

where $\boldsymbol{s} = x\boldsymbol{r} + \boldsymbol{r'}$. Using the fast-decoding property of E (property 3), Bob recovers the vector $x\boldsymbol{a} + \boldsymbol{b}$ (by making $O(k^3)$ arithmetic operations) and outputs $x\boldsymbol{a} + \boldsymbol{b}$.

We are now ready to show that the reverse vector OLE protocol works:

Lemma 1. *Suppose that Assumption 2 holds. Then Construction 1 implements the Reverse Vector-OLE functionality of width $w = \Theta(k)$ over $\mathbb{F}$ with semi-honest and computational security in the OT-hybrid model. Furthermore, ignoring the cost of initialization, the arithmetic complexity of the protocol is $O(w)$.*

Proof. The running time follows easily by inspection of the protocol. We prove correctness. By Assumption 2 (property 3), except with negligible probability Bob recovers the vector $\boldsymbol{s}$ correctly. Also, by a Chernoff bound, the v-suffix of the error vector $\boldsymbol{e}$ contains at most $\mu' v$ non-zero coordinates. Therefore, the decoding procedure of the error-correcting code succeeds.

As for privacy, consider first the case where Alice is corrupt. We can then simulate Bob's message with a random vector in $\mathbb{F}^{u+v}$ which will be computationally indistinguishable by Assumption 2. If Bob is corrupt, we can simulate what Bob receives in OTs given Bob's output $x\boldsymbol{a} + \boldsymbol{b}$, namely we compute $\boldsymbol{f} = E_{\boldsymbol{s}}(x\boldsymbol{a} + \boldsymbol{b})$ for a random $\boldsymbol{s}$ and sample a set I as in the protocol (each coordinate $i \in [k^3]$ is chosen with probability $1 - \mu$). Then for the OT in position i, we let Bob receive $\boldsymbol{f}_i$ if $i \in I$ and nothing otherwise. This simulates Bob's view perfectly, since in the real protocol $\boldsymbol{s} = x\boldsymbol{r} + \boldsymbol{r'}$ is indeed uniformly random, and the received values for positions in I do not depend on x or $\boldsymbol{e}$, only on $\boldsymbol{s}$ and Bob's output. □

3.3 From reverse vector-OLE to vector-OLE

Finally, to get a protocol for the vector OLE Functionality, note that we can get such a protocol from the Reverse vector OLE functionality:

Construction 2 (vector-OLE Protocol) *Given an input $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^w$ for Bob, and $x \in \mathbb{F}$ for Alice, the parties do the following:*

1. *Call the Reverse Vector-OLE functionality, where Bob uses input $\boldsymbol{a}$ and Alice uses input x and a randomly chosen $\boldsymbol{b'} \xleftarrow{\$} \mathbb{F}^w$. As a result, Bob will receive $x\boldsymbol{a} + \boldsymbol{b'}$.*
2. *Bob sends $\boldsymbol{b} + (x\boldsymbol{a} + \boldsymbol{b'})$ to Alice. Now, Alice outputs $(\boldsymbol{b} + (x\boldsymbol{a} + \boldsymbol{b'}) - \boldsymbol{b'} = x\boldsymbol{a} + \boldsymbol{b}$.*

It is trivial to show that this implements the vector-OLE functionality with perfect security. Combining the above with Lemma 1, we derive the following theorem.

Theorem 3. *Suppose that Assumption 2 holds. Then, there exists a protocol that implements the vector-OLE functionality of width w over $\mathbb{F}$ with semi-honest computational security in the OT-hybrid model with arithmetic complexity of $O(w) + \mathrm{poly}(k)$.*

Proof. For $w < k^3$, the theorem follows directly from Construction 2 and Lemma 1 (together with standard composition theorem for secure computation). The more general case (where w is larger) follows by reducing long w-vector OLE's into t calls to w_0-vector OLE for $w_0 = \Theta(k^3)$ and $t = w/w_0$. Since initialization is only performed once (with a one-time $\mathrm{poly}(k)$ cost) and M is re-used, the overall complexity is $\mathrm{poly}(k) + O(tw_0) = \mathrm{poly}(k) + O(w)$ as claimed. □

Remark 1 (Implementing the OTs). First, note that the OT variant we need can be implemented efficiently for large fields as follows: Alice chooses a short *seed* for a PRG and to send field element f, she sends $f \oplus PRG(seed)$ and then does an OT where she offers Bob *seed* and a random value. If Bob wants to receive f, he chooses to get *seed*, otherwise he choose the random value.

Our protocol employs $O(w)$ such OTs on field elements, or equivalently, on strings of length $\log |\mathbb{F}|$ bits. For sufficiently long strings (i.e., $w = \mathrm{poly}(k)$ for sufficiently large polynomial) one can get these OT very cheaply both practically and theoretically.

Indeed, the implementation we described (which is similar to an observation from [35]), can be done with optimal asymptotic complexity of $O(w \log |\mathbb{F}|)$ bit operations assuming the existence of a linear-stretch

pseudorandom generator $G : \{0,1\}^k \to \{0,1\}^{2k}$ which is computable in linear-time $O(k)$. Moreover, such a generator can be based on the binary version of Assumption 2, as follows from [9]. In practice, we can get the OT's very efficiently via OT extension and perhaps (for very large fields) using a PRG based on AES which is extremely efficient on modern Intel CPUs.

Remark 2 (On the achievable rate). Note that the full vector OLE protocol communicates $u+v$ field elements, does $u+v$ OTs and finally sends w field elements. The rate is defined as the size of the output (w) divided by the communication complexity. Now, asymptotically, we can ignore u since it is $o(v)$. Furthermore, v is the length of the code Ecc, which needs to be about $w/(1-\mu)$ to allow for erasure decoding w values from a fraction of μ random erasures. By the previous remark, an OT can be done at rate 1, so it counts as 1 field element. So we find that the asymptotic rate approaches $(1-\mu)/(3-\mu)$ (e.g., $3/11 \approx 1/4$ for $\mu = 1/4$). If Assumption 2 holds for any constant error rate $\mu > 0$ then we can obtain rate approaching $1/3 - \epsilon$ for any constant $\epsilon > 0$. Furthermore, making Assumption 2 for, say, $\mu(k) = 1/\log_2 k$, gives us an asymptotic rate of $1/3$ (i.e., the ratio between w and the number of communicated field elements tends to $1/3$ as w and k tend to infinity). Note that in the context of binary codes, LPN-style assumptions with sub-constant μ are quite standard (for instance, such assumptions underly Alekhnovich's construction of public-key encryption from LPN [1]).

4 Batch-OLEs

In this section we implement n copies of OLE (of width 1) with constant computational overhead based on vector-OLE with constant computational overhead and a polynomial-stretch arithmetic pseudorandom generator of constant depth. The transformation is similar to the one described in [35] for the binary setting, and is based on a combination Beaver's OT extension [14] with a decomposable randomized encoding.

4.1 From vector-OLE to $\mathbf{NC^0}$ functionalities

We begin by observing that local functionalities can be reduced to vector-OLE with constant computational overhead. This follows from an arithmetic variant of Yao's protocol [55] where the garbled circuit is replaced with fully-decomposable randomized encoding. For simplicity, we restrict our attention to functionalities in which only the first party Alice gets the input.

Lemma 2. *Let $\mathbb{F}$ be a finite field and let f be a two-party $\mathbf{NC^0}$ functionality that takes ℓ_1 field elements from the sender, ℓ_2 field elements from the receiver, and delivers m field elements to the receiver. Then, we can securely compute f with an information-theoretic security in the semi-honest model with arithmetic complexity of $O(m)$ and by making $O(\ell_2)$ calls to ideal $O(m/\ell_2)$-width OLE.*

The constant in the big-O notation depends on the circuit depth of f.

Proof. View f as a function over $\mathbb{F}^\ell$ where $\ell = \ell_1 + \ell_2$. By Fact 1, there exists a DARE $\hat{f}$ which can be encoded and decoded by an $O(m)$-size arithmetic circuit. Recall, that

$$\hat{f}(x;r) = (\hat{f}_0(r), (\hat{f}_i(x_i;r))_{i\in[\ell]}), \qquad \text{where } \hat{f}_i(x_i;r) = x_i \boldsymbol{a}_i(r) + \boldsymbol{b_i}(r).$$

Since the encoding is computable by $O(m)$-size circuit, it is also possible to take r and collectively compute $(\boldsymbol{a}_i(r), \boldsymbol{b}_i(r))_{i\in[\ell]}$ by $O(m)$ arithmetic operations. Also, the total length of these vectors is $O(m)$.

Let us denote by $A \cup B$ the partition of $[\ell]$ to the inputs given to Alice and the inputs given to Bob, and so $|A| = \ell_1$ and $|B| = \ell_2$. Let $w = m/\ell_2$ and assume an ideal vector OLE of width w. Given an input x_A for Alice and x_B for Bob, the parties use Yao's garbled-circuit protocol to compute f as follows:

- Bob selects randomness $r \xleftarrow{\$} \mathbb{F}^\rho$ for the encoding and sends $\hat{f}_0(r)$ together with $(\hat{f}_i(x_i;r))_{i\in B}$.

– For every $i \in A$ the parties invoke width w-OLE where Alice uses x_i as her input and Bob uses $(\boldsymbol{a}_i(r), \boldsymbol{b}_i(r))$ as his inputs. If the length W_i of $\boldsymbol{a}_i(r)$ and $\boldsymbol{b}_i(r)$ is larger than w, the vectors are partitioned to w-size blocks and the parties use $\lceil (W_i/w) \rceil$ calls to w-width OLE. (In the j-th call Bob uses the j-th block of $(\boldsymbol{a}_i(r), \boldsymbol{b}_i(r))$ as his input and Alice uses x_i as her input.)
– Finally, Alice aggregates the encoding $\hat{f}(x; r)$, applies the decoder and recovers the output $f(x)$.

It is not hard to verify that both parties can be implemented by making at most $O(\ell)$ arithmetic operations. (In fact, they can be implemented by $O(\ell)$-size arithmetic circuits). Moreover, the number of call to the w vector-OLE is $\sum_{i \in A} \lceil W_i/w \rceil = O(m/w) = O(\ell_2)$. The correctness of the protocol follows from the correctness of the DARE. Assuming perfect OLE, the protocol provides perfect security for Bob (who gets no message during the protocol) and for Alice (whose view can be trivially simulated using the perfect simulator of the DARE). □

4.2 From pseudorandom-OLE to OLE

The following lemma is an arithmetic variant of Beaver's reduction from batch-OT to OT with "pseudorandom" selection bits.

Lemma 3. *Let $G : \mathbb{F}^k \to \mathbb{F}^n$ be a pseudorandom generator. Consider the two-party functionality g that takes a seed $\boldsymbol{s} \in \mathbb{F}^k$ from Alice and n pairs of field elements $(a_i, b_i), i \in [n]$ from Bob and delivers to Alice the value $y_i a_i + b_i$ where $\boldsymbol{y} = G(\boldsymbol{s})$. Then, in the g-hybrid model it is possible to securely compute n copies of OLE of width 1 with semi-honest computational security and complexity of $O(n)$ arithmetic operations and a single call to g.*

Proof. Let $\boldsymbol{x} = (x_i)_{i \in [n]}$ be Alice's input and let $(a_i, b_i), i \in [n]$ be Bob's input.

1. Alice and Bob call the protocol for g where Alice uses a random seed $s \xleftarrow{\$} \mathbb{F}^k$ as an input and Bob uses the pairs $(a_i, c_i), i \in [n]$ where $c_i \xleftarrow{\$} \mathbb{F}$ are chosen uniformly at random. Alice gets back the value $u_i = y_i a_i + c_i$ for $i \in [n]$.
2. Alice sends to Bob the values $\Delta_i = x_i - y_i$, for every $i \in [n]$.
3. Bob responds with $v_i = \Delta_i a_i + (b_i - c_i)$ for every $i \in [n]$.
4. Alice outputs $z_i = u_i + v_i$ for every $i \in [n]$.

It is not hard to verify that correctness holds, i.e., $z_i = x_i a_i + b_i$. To prove security, observe that Alice's view, which consists of $(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v})$, can be perfectly simulated. Indeed, given an input $\boldsymbol{x}$ and an output $\boldsymbol{z}$: Sample $\boldsymbol{s} \xleftarrow{\$} \mathbb{F}^k$ together with $\boldsymbol{u} \xleftarrow{\$} \mathbb{F}^n$ and set $\boldsymbol{v} = \boldsymbol{z} - \boldsymbol{u}$. As for Bob, his view consists of $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ and a pseudorandom string $\boldsymbol{\Delta}$. We can therefore simulate Bob's view by sampling $\boldsymbol{\Delta}$ (and $\boldsymbol{c}$) uniformly at random. □

4.3 From $\mathbf{NC^0}$ PRG to batch-OLE

To get our final result, we need a polynomial-stretch $\mathbf{NC^0}$ arithmetic pseudorandom generator. In fact, it suffices to have a collection of such PRG's.

Assumption 4 (polynomial-stretch $\mathbf{NC^0}$ PRG (arithmetic version)) *There exists a polynomial-time algorithm that given 1^k and a field representation $\mathbb{F}$ samples an $\mathbf{NC^0}$ mapping $G : \mathbb{F}^k \to \mathbb{F}^{k^2}$ (represented by a circuit) such that with all but negligible probability G is a pseudorandom generator against* $\mathrm{poly}(k)$ *adversaries.*

Assumption 4 is discussed in Section 7. For now, let us mention that similar assumptions were made in the binary setting and known binary candidates have natural arithmetic variants.

Combining Lemmas 2 with 3, we get the following theorem.

Theorem 5. *Suppose that Assumption 4 holds. Then, it is possible to securely compute n copies of OLE over $\mathbb{F}$ in the semi-honest model by making $O(n/k)$ calls to ideal $O(k)$-width OLE and $O(n) + \mathrm{poly}(k)$ additional arithmetic operations.*

Proof. Let $t = n/k^2$. Implement the OLE's using t batches each of size k^2. By Lemmas 2 and 3, each such batch can be implemented by making k calls to ideal $O(k)$-width OLE and $O(k^2)$ additional arithmetic operations. Since the initialization of the pseudorandom generator has a one-time poly(k) cost, we get the desired complexity. □

Combining Theorems 3 and 5, together with an optimal OT implementation (which by Remark 1 follows from standard OT), and plugging in standard composition theorem for secure computation, we derive the following theorem.

Corollary 1 (main result). *Suppose that Assumptions 2 and 4 hold, and a standard binary OT exists. Then, there exists a protocol for securely computing n copies of OLE over $\mathbb{F}$ with semi-honest computational security, and arithmetic complexity of $O(n) + \text{poly}(k)$.*

5 Applications of Vector-OLE

In the previous section we used vector-OLE only as a tool to obtain OLE. However, there are applications where vector-OLE is precisely what we need.

First, it is easy to see that a secure multiplication of an $n \times n$ matrix by a length-n vector reduces to n instances of width-n vector-OLE. Therefore, using our implementation of vector-OLE, it is straightforward to multiply a matrix by a vector with with $O(n^2)$ field operations, which is asymptotically optimal, and with a small concrete overhead. This can be used as a building block for other natural secure computation tasks, such as matrix multiplication and other instances of secure linear algebra; see [18, 45] for other examples and motivating applications.

Another class of applications is where a party holds some object that needs to be compared to entries in a database held by another party. The characteristic property is that the input of party is fixed whereas the input from the other party varies (as we run through the database). A good example is secure face recognition, where a face has been measured in some location and we now want to securely test if the measurement is close to an object in a database – containing, say, suspects of some kind. This reduces to computing the Euclidean distance from one point in a space of dimension m (say) to n points in the same space, and then comparing these distances, perhaps to some threshold. It is clearly sufficient to compute the square distance, so this means that what we need to compute will numbers of form

$$\sum_i (x_i - y_i^j)^2 = \sum_i x_i^2 + (y_i^j)^2 - 2x_i y_i^j,$$

where $(x_1, ..., x_m)$ is the point held by the client, and $(y_1^j, ..., y_m^j)$ is the $j'th$ point in the database. Clearly, additive shares of x_i^2 and $(y_i^j)^2$ can be computed locally, while additive shares of $2x_i y_i^j$ can be done using vector-OLE, namely we fix i and compute $2x_i \cdot (y_i^1, ..., y_i^n)$.

Once we have additive shares of the square distances, the comparisons can be done using standard Yao-garbling. Since this only requires small circuits whose size is independent of the dimension m, this can be expected to add negligible overhead.

We note that the secure face recognition problem was considered in [22], where a solution based on Paillier encryption was proposed (see [51] for optimizations). This adds a very large computational overhead compared our solution, since an exponentiation is required for each product $2x_i y_i^j$.

Similar applications of vector-OLE can apply in many other contexts of securely computing on numerical data that involve computations of low-degree polynomials. See, e.g., [17, 25] and references therein for some recent relevant works in the context of secure machine learning.

6 Implementation

We have implemented the vector-OLE protocol. This is the most practical of our constructions and, as we explained in the previous section, it has applications of its own, even without the conversion to OLEs of width 1.

6.1 Choice of the matrix M

For the vector OLE protocol, we need a fast pseudorandom matrix M (see Assumption 2). For this, we have chosen to use a random d-sparse matrix for a suitable constant d. This means we are basing ourselves on Assumption 6 from Section 7, which (together with Lemma 4) essentially implies that a random d-sparse matrix is likely to satisfy some combinatorial properties (good "expansion" and "dual distance") which leads to pseudorandomness (i.e., satisfy Assumption 2).

Our parameters are based on the best known distinguishing attack from [56] whose complexity is exponential in $H_2(\mu)t$ where μ is the noise rate, $H_2(\cdot)$ is the binary entropy function, and t is the size of the smallest set S of rows in M that have a joint support of $t(1-\mu)$ (i.e., S is "shrinking" by a factor of $(1-\mu)$.) Correspondingly, to get b bits of security, we select the size of M, such that, except with tiny probability, every set S of at most $b/H_2(\mu)$ rows has a support of at least $(1-\mu)|S|$. This level of expansion is somewhat optimistic, but still seems to defend against the best known attacks.[12] The choice of noise rate is a tradeoff: if it is too small we have to increase the security parameters, if it is too large the (communication) rate will be bad. We chose noise rate $1/4$ because it allows communication rates up to $1/5$ as we shall see later and also allows us to be secure against known attacks with reasonably small parameters for the matrix size.

In the earlier theory sections we have assumed that the number of rows in M is $\Theta(k^3)$. This was because we wanted to amortize away the $O(k^3)$ amount of work needed to do Gaussian elimination using the top part of the matrix. However, to achieve this number of rows in the concrete security analysis we would need to go to rather large values of k, and this would create some issues with memory management. Hence, to get a more practical version with a relatively small footprint, we chose to settle for $O(k^2)$ rows. Then, for 80-bit security and $d = 10$ it turns out that we will need approximately $k = 182$ columns and k^2 rows, while for 100-bit security we need $k = 240$.

Note that once the number of rows and columns is fixed, this also fixes the parameters u, v from the vector OLE protocol.

6.2 ECC: Using Luby Transform Codes

It remains to consider the erasure correcting code ECC. For this, we want to use Luby Transform (LT) codes [43]. LT codes have extremely simple and efficient en- and decoding algorithms, using only field addition and subtraction, no multiplications or inversions are needed. On the other hand, LT codes were designed for a streaming scenario, where one continues the stream until the receiver has enough data to decode. In our case, we must stop at some finite codeword size, and this means we will have a non-negligible probability that decoding fails. In practice, one can think of this as a small but constant error probability, say 1%. On the other hand, this be detected, and the event that decoding fails only depends on the concrete choice of LT code and the choice of the noiseless positions.

Since the player A knows the LT code to be used and is also the one who chooses the noise pattern, he can simply choose a random noise pattern subject to the condition that decoding succeeds.

The protocol will then always terminate successfully, but we need to make a slightly stronger computational assumption to show that the protocol is secure: the pseudorandomness condition for the matrix M must hold even if we exclude, say 1% of the possible noise patterns. It turns out that, given the known attacks, excluding any 1% of the noise patterns makes no significant difference.[13]

More concretely we instantiate the encoding function $\mathsf{Ecc} : \mathbb{F}^w \to \mathbb{F}^v$ over the *Robust Soliton distribution* also defined in [43]. One generates a output symbol by sampling a degree *dec* from that distribution and

[12] More conservative choices lead to provable security against large families of attacks, however, we believe that the security proofs may not be tight. See the discussion in Section 7 and in [56].

[13] Indeed, since we remove a *small* subset of all possible noise patters, the remaining patterns cannot be linearized, i.e., cannot be written as a low-degree function of few fresh variables, and so known attacks do not seem to apply. Of course, one should make sure that the excluded noise patterns do not correlate somehow with the choice of the "pseudoranodm" matrix M (say in a way that leaves few "special" coordinates of the secret random seed, r, uncovered). However, in our case, the matrix M is chosen at random independently of the choice of the LT-code (which determines the excluded noise patterns). See also the discussion in Section 7.

defining the symbol as the sum of *dec* input symbols chosen uniformly among alle the input symbols. This distribution is defined over two constant parameters δ and c. Here δ denotes the probability of failed decoding, which together with c adds extra weight to the probability of smaller degree encoding symbols. The two parameters also determine a constant β for which $v = w\beta$, but since v and w is fixed in our construction, β is also fixed, and we have one degree of freedom less. Thus we instantiate the distribution with parameters w, v and δ and let those determine c such that $\beta = v/w$.

Note that δ may deviate from the actual probability of failed decoding λ depending on the concrete code. We estimate λ by testing our code on $50,000$ random codewords. Note that we fixed the value of v earlier, as a result of choosing M. Given this, we tested different combinations of w and δ to achieve a code decodes $w/4$ errors with probability λ. Our concrete parameters are shown in Table 1. Here is presented different choices for w and δ that shows how one may trade width for failure probability. In the implementation we will use the codes corresponding to $\delta = 0.01$ for both security parameters.

Table 1: Implementation parameters

k	u	v	w	δ	λ
182	244	$33,124$	5.000	0.001	0.0017
			10,000	0.01	0.016
			14,000	0.1	0.095
240	320	$57,600$	10.000	0.001	0.0003
			20,000	0.01	0.015
			23,000	0.1	0.069

6.3 Doing Oblivious Transfers

In the vector OLE protocol we need 1 OT for each row of M. It is natural to implement this via OT extension which can be done very efficiently in a situation like ours where we need a substantial number of OTs. For instance, in [37, 13], an amortized time of about 0.2 μs per semi-honestly secure string OT was obtained, when generating enough of them in one go. Note that in the protocol specification, we required a special OT variant where one message is sent and the receiver chooses to get it or not. But this can of course be implemented using standard 1-2 string OT where the sender offers the message in question and a dummy.

In order to not require a specific relation between the number of OTs produced by one run of an OT extension and what our protocol requires, we have assumed that we precompute a number of random OTs, which we then adjust to the actual values using standard techniques. The adjustment requires one message in both directions where the first one can be sent in parallel with the message in the Vector OLE protocol, so we get a protocol with a total of 3 messages.

We have not implemented the OT extension itself, instead we simulate the data and communication needed when using the preprocessed OTs. The hypothesis is that that time required to create the random OTs in the first place is insignificant compared to the rest of the computation required. We discuss below the extent to which this turned out to be true.

6.4 Communication Overhead

Having fixed the parameter choices, we can already compute the communication we will need: we can ignore the communication relating to the top part of the matrix M as this is responsible for less than 1% of the communication. Then, by simple inspection of the protocol, one sees that we need to send $v + w$ field element and do v OTs. We implement the OTs directly from 1-2 OT which means an OT costs communication of 2

field elements and 1 bit. So we get a total of $3v + w$ field elements (plus v bits, which we can ignore when the field is large). With our choice of LT code, v is roughly $3w$, so we have $10w$ field elements to send. Hence the rate is indeed constant, as expected, namely 1/10. Accepting a larger failure probability for LT decoding, we could get a rate of roughly 1/7. As explained in Remark 2, the best we can hope for asymptotically is about 1/4 when the noise rate is 1/4.

There are two reasons why we do not reach this goal: first, we chose to use LT codes for erasure correction to optimize the computational overhead, but this comes at the price of a suboptimal rate. Second we implement the OTs at rate 1/2. As explained in Remark 1, rate (almost) 1 is possible, but only for large fields. So for fields of size 1000 bits or more, we believe the rate of our implementation can be pushed to about 1/5 without significantly affecting its concrete computational overhead.

6.5 Test Set-up and Results

Our set-up consists of two identical machines, each with 32GB RAM and a 64-bit i7-3770K CPU running at 3.5GHz. The machines are connected on a 1GbE network with $0.15ms$ delay.

A b-bit field is instantiated by chosing $\mathbb{F}_p$ for the largest prime $p < 2^b$. All matrix operations are optimized to that of sparse matrices except for the Gaussian elimination, where we construct an augmented matrix and do standard row reduction. All parameters are loaded into memory prior to the protocol execution including the matrix M, the LT code and a finite set of test vectors.

First a version is implemented using the GNU Multiple Precision Arithmetic Library for finite field arithmetic. We benchmark this version with b-bit field for $b \in \{32, 64, \ldots, 2048\}$. In this setting we allocate $2b$ bits for each element once, such that we never have to allocate more e.g. at multiplication operations, which consists of a MUL and MOD GMP call. We further replace the MOD call after addition and subtraction with a conditional sum.

Since most computation in the protocol includes field operations, we optimized the finite field for 32-bit and 64-bit versions. Here the 32-bit version only use half of the machine's word size, but offers fast modulo operation after a multiplication with the DIV instruction. The 64-bit version utilizes the full word size, but relies on the compiler's implementation of the modulo operation for UINT128_T as supported in GCC-based compilers. For random number generation, we use the Mersenne Twister SFMT variant instead of GMP.

In Table 2 and Table 3 it is shown how the GMP and the optimized version compare for respectively $k = 182$ and $k = 240$. Here, we measure the amortized time per single OLE, or more precisely, since the protocol securely computes the multiplication of a scalar by a vector of length w, we divide the time for this by w to get the time per oblivious multiplication. We obtain these times by having as many threads as possible run the protocol in a loop and counting only successful executions. These amortized timings are also depicted in Figure 1. Afterwards we run the protocol sequentially in a single thread and measure how fast we can execute one instance of the protocol. This indicates the latency, i.e., the time taken from the protocol starts until data is ready. Finally, since we use much less network speed than what is available, we present the network bandwidth we actually use, as this may become a limiting factor in low-bandwidth networks. The reason why the optimized versions use more bandwidth than corresponding GMP versions is that they are computationally faster, so the network is forced to handle the same amount of communication in shorter time. Then for larger fields, bandwidth usage increases because larger field elements need to be sent, but for the largest field size (2048 bits) we see a decrease because computation now has slowed down to the extent that there is more than twice the time to send field elements of double size (compared to 1024 bits).

We did not list the communication complexity, but this is easily computed as $n/r = 10n$ where n is bit size of field elements and $r = 1/10$ is the rate.

We note the protocol latency for 100-bit security is about 2-3 times that of 80-bit security. But for the amortized times the increase in security parameter comes cheaply because we double w in going from 80 to 100-bit security.

In our setup, we need to execute between 2 and 3 OTs per single OLE. Given the results from [13] which were obtained on an architecture similar to ours, we can expect these to take an amortised time of 0.6 μs, which as expected becomes insignificant as the field size grows, but cannot be ignored for the optimized version on smaller fields.

Table 2: Benchmark of the Vector-OLE protocol for $k = 182$ and $w = 10,000$

Field size	**Version**	**time per OLE**	**Latency**	**Consumed Bandwidth**
32 bit	Optimized	$0.56\mu s$	$0.04s$	45.53 MB/s
64 bit	Optimized	$1.00\mu s$	$0.14s$	50.83 MB/s
32 bit	GMP	$3.65\mu s$	$0.26s$	6.98 MB/s
64 bit	GMP	$3.66\mu s$	$0.27s$	13.92 MB/s
128 bit	GMP	$4.24\mu s$	$0.31s$	24.03 MB/s
256 bit	GMP	$6.37\mu s$	$0.47s$	31.98 MB/s
512 bit	GMP	$9.58\mu s$	$0.64s$	42.50 MB/s
1024 bit	GMP	$18.29\mu s$	$1.15s$	44.53 MB/s
2048 bit	GMP	$50.85\mu s$	$2.87s$	32.04 MB/s

Table 3: Benchmark of the Vector-OLE protocol for $k = 240$ and $w = 20,000$

Field size	**Version**	**time per OLE**	**Latency**	**Consumed Bandwidth**
32 bit	Optimized	$0.70\mu s$	$0.12s$	31.70 MB/s
64 bit	Optimized	$1.14\mu s$	$0.25s$	38.86 MB/s
32 bit	GMP	$3.96\mu s$	$0.48s$	5.57 MB/s
64 bit	GMP	$3.97\mu s$	$0.48s$	11.12 MB/s
128 bit	GMP	$4.52\mu s$	$0.56s$	19.56 MB/s
256 bit	GMP	$6.61\mu s$	$0.82s$	26.75 MB/s
512 bit	GMP	$9.93\mu s$	$1.15s$	35.59 MB/s
1024 bit	GMP	$19.48\mu s$	$2.22s$	36.29 MB/s
2048 bit	GMP	$51.73\mu s$	$5.45s$	27.34 MB/s

As computation is the bottleneck compared to network bandwidth, we identify which part of the computation is the most expensive. We test the optimized 32-bit version for $k = 182$ and focus on the Gaussian elimination, the Luby encoding and decoding and a matrix-vector product $c = M \cdot r$. This is presented in Table 4 as an index set. Here the Gaussian elimination acts as base value and takes 45% of the total protocol time including communication.

Table 4: Timing of computation

Operation	**Index**
Gaussian elimination	100
Luby decoding	22
Luby encoding (Ecc)	3
Encode $c = M \cdot r$	13

Since the Gaussian elimination costs more than other parts of the protocol, this means that one would need to increase w for the amortization to work. However one could replace this step with any algorithm for solving linear systems, in particular algorithms taking advantage of matrix sparsity such as [54]. Finally one may take advantage of specific constructions of finite fields allowing for even faster arithmetic operations.

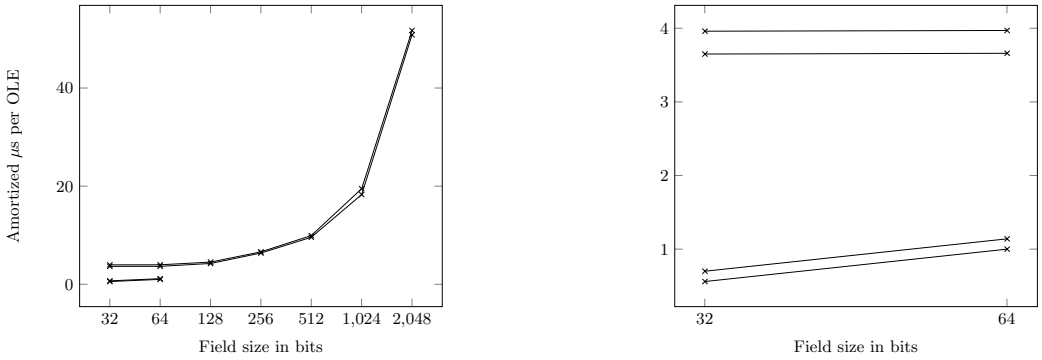

Fig. 1: Amortized time per OLE compared to field size

7 About the Assumptions

Our results rely on two types of assumptions, both of which can be viewed as natural arithmetic analogues of assumptions that have been studied in the boolean case. We discuss our instantiations of these assumptions below. In Section 7.1 we discuss the assumption we use for instantiating our constant-overhead vector-OLE protocol, whereas in Section 7.2 we discuss the additional assumption used for obtaining constant-overhead protocol for general arithmetic computations.

7.1 Instantiating Assumption 2 (Fast pseudorandom matrix)

An distribution ensemble $\mathcal{M} = \{\mathcal{M}_k\}$ over $m(k) \times k$ matrices is pseudorandom for noise rate μ if it satisfies property 2 of Assumption 2. It is natural to assume that, for every $m = \text{poly}(k)$, a random $m \times k$ matrix is pseudorandom over any finite field. (This is the arithmetic analogue of the Decisional-Learning-Parity-with-Noise assumption [30, 15, 50]). However, Assumption 2 requires the corresponding linear map to be computable in $O(m)$ arithmetic operations (together with an additional linear independence condition). We suggest two possible instantiations for this assumption.

The Druk-Ishai Ensemble. Druk and Ishai [21] constructed, for any finite field $\mathbb{F}$ and any code length $m \in \text{poly}(k)$, a probabilistic ensemble $\mathcal{M}$ of linear-time computable (m, k) error-correcting code over $\mathbb{F}$ whose minimal distance is close to the Gilbert-Varshamov bound [26, 53] with overwhelming probability. It was further conjectured that, over the binary field, the ensemble is pseudorandom for arbitrary polynomial $m(k)$.[14] The assumption seems to hold for arbitrary finite fields as well. Moreover, the ensemble satisfies Condition 3 of Assumption 2 since, by [21, Theorem 5], every subset of $m' = \omega(k)$ rows of the code generates, except with negligible probability, a code of distance $1 - 1/|\mathbb{F}| - o(1)$.

Alekhnovich's Ensemble. Alekhnovich [1] conjectured that a random d-sparse binary matrices (that each of its rows contain exactly d non-zero elements) is likely to be pseudorandom for constant noise rate. A closer examination of this assumption [6] suggests that pseudorandomness holds for any sparse matrix with sufficiently large *dual distance*, where the dual distance of a matrix M, denoted by $\text{dd}(M)$, is the maximal integer D for which every subset of M's rows of size at most D is linearly independent over $\mathbb{F}$.[15] We will use the arithmetic version of this assumption.

[14] The basic construction is described for codes with codeword of length $m = O(k)$; however, one can extend it for codes with codeword of polynomial length $m(k)$, by independently sampling polynomially many $O(k) \times k$ generating matrixes and placing them one on top of the other to get a $\text{poly}(k) \times k$ matrix. The pseudorandomness assumption of [21, Section 5.1] applies to this variant for arbitrary polynomial number of samples.

[15] The name "dual distance" comes from the fact that the left null space of M is a linear code of distance $\text{dd}(M)$.

Assumption 6 (Arithmetic version of Alekhnovich's assumption) *For every prime-order field $\mathbb{F}$ every polynomial $m(k)$, every constant d, every real $\mu \in (0, 1/2)$, and every d-sparse matrix $M \in \mathbb{F}^{m\times k}$, the following holds. Any circuit of size $T = \exp(\Omega_\mu(\mathrm{dd}(M)))$ cannot distinguish with advantage better than $1/T$ between the uniform distribution $\boldsymbol{v} \xleftarrow{s} \mathbb{F}^m$ to the distribution $(M\boldsymbol{r} + \boldsymbol{e})$, where $\boldsymbol{r} \xleftarrow{s} \mathbb{F}^k$ and $\boldsymbol{e} \xleftarrow{s} \mathcal{D}_\mu(\mathbb{F}_p)^\ell$.*

The assumption says that the level of security is exponential in the dual distance where the hidden constant in the exponent may depend on the noise rate μ. As we will see below for a properly chosen d-sparse matrix M of dimensions $\mathrm{poly}(k)\times k$, we can expect a dual distance of k^ε, and so we get sub-exponential security.[16] Assumption 6 is consistent with the best known attacks, and, can be analytically established for a large family of algorithms including myopic algorithms, linear-tests, low-degree polynomials, constant depth circuits and product tests (see Zichron's Master thesis [56]). Also observe that since the matrix M is d-sparse, the linear mapping $f_M : \boldsymbol{x} \mapsto M\boldsymbol{x}$ can be computed by performing $O(dm) = O(m)$ arithmetic operations.

How to sample matrices with large dual distance? We suggest to sample a d-sparse matrix $M \in \mathbb{F}^{m\times k}$ in two steps. First, choose the locations of the non-zero entries of the matrix (e.g., by selecting a random set of d entries per row), and then fill them with random field elements. The outcome of the first step can be viewed as a d-sparse $m \times k$ zero-one matrix G. To analyze the process, we relate the dual distance of the final matrix M to the expansion properties of the matrix G which can be naturally viewed as a d-uniform hypergraph over the vertex set $[k]$ with m hyperedges. (Hereafter referred to as (m, k, d)-hypergraph.)

It is well known that if every set S of at most r hyperedges in G expands by a factor $\alpha > d/2$ (i.e., the hyperedges in S "touch" more than $\alpha|S|$ vertices) then M will have (with probability 1) a dual distance of at least r. Interestingly, we show that, over large field $\mathbb{F}$, it suffices to require a much weaker expansion factor of $(1 + \varepsilon)$ that is independent of the sparsity parameter d. This is essentially optimal since a shrinking set of hyperedges (which expands by a factor smaller than 1) induce a linearly-dependent set of rows in M. By analyzing the expansion of a random sparse hypergraphs (using standard tools) we derive estimation for the dual distance of M (which are better than the ones available for small fields).

Formally, let us denote by $\mathcal{M}(G, \mathbb{F})$ the outcome of the second step of the process applied to some (m, k, d)-hypergraph G, and let $\mathcal{M}(m, k, d, \mathbb{F})$ denote the distribution obtained by selecting the (m, k, d)-hypergraph G at random and then sampling from $\mathcal{M}(G, \mathbb{F})$. In particular, the following lemma is proved in [56].

Lemma 4. *Suppose that G is a (m, k, d)-hypergraph which is $(r, 1 + \varepsilon)$-expanding and $|\mathbb{F}|^\varepsilon > m$. Then,*

$$\Pr_{M\leftarrow\mathcal{M}(G,\mathbb{F})}[\mathrm{dd}(M) < r] < |\mathbb{F}|^{-1} \sum_{t=1}^{r} \left(\frac{m}{|\mathbb{F}|^\varepsilon}\right)^t.$$

Consequently, a random $M \leftarrow \mathcal{M}(m, k, d, \mathbb{F})$, with $|\mathbb{F}|^\varepsilon > m$ and $m = \Delta k$ has dual-distance of $r = \frac{k}{\Delta^{\frac{1}{d-2.1}}}$ with probability of at least $1 - O(|\mathbb{F}|^{-1}) - o(1)$.

Remarks:

1. (Linear Independence) Recall that Assumption 2 requires that a random subset of $k \log^2 k$ of the rows of M have, except with negligible probability, full rank. In Lemma 6 we show that this condition holds as long as G is *semi-regular* in the sense that each of its nodes participates in at least $\Omega(m/k)$ hyperedges.
2. (Different noise distributions) The choice of i.i.d based noise is somewhat arbitrary and it seems likely that other noise distributions can be used. In fact, it seems plausible that one can use any noise distribution which has high entropy and cannot be approximated by a low-degree function of few fresh variables (and thus is not subject to linearization attacks such as the ones from [12]).

Given the above discussion, Assumption 2 now follows from Assumption 6 and the existence of an explicit family of expanders.[17] The latter point is discussed in Section 7.2.

[16] An exponential level of security can be achieved only when the number of rows is linearly larger than the number of columns.

[17] Indeed, in the conference version of the paper, Assumption 6 was stated directly in terms of expansion. We believe that the current version (which implies the previous version) is more informative.

7.2 Instantiating Assumption 4 ($\mathbf{NC^0}$ polynomial-stretch PRG)

In the binary setting, the existence of locally-computable polynomial-stretch PRG was extensively studied in the last decade. (See [4] and references therein.) Let $f : \mathbb{F}^k \to \mathbb{F}^m$ be a d-local function which maps a k-long vector x into an m-long vector $(P_1(x_{S_1}), \ldots, P(x_{S_m}))$ where $S_i \in [k]^d$ is a d-tuple and P_i is a d-variate multi-linear polynomial. Over the binary field, it is conjectured that as long as the (k, m, d) hypergraph $G = (S_1, \ldots, S_m)$ is expanding and the P_i's are sufficiently "non-degenerate" the function forms a good pseudorandom generator. (This is an extension of Goldreich's original one-wayness conjecture [29].) In fact, this is conjectured to be the case even if all the polynomials $P_1, \ldots, P_m$ are taken to be the same polynomial P. We denote the resulting function by $f_{G,P}$ and make the analog arithmetic assumption. In the following we say that a function $f : \mathbb{F}^k \to \mathbb{F}^m$ is T-pseudorandom if every circuit of size at most T cannot distinguish $f(x), x \xleftarrow{\$} \mathbb{F}^k$ from $y \xleftarrow{\$} \mathbb{F}^m$ with advantage better than $1/T$.

Assumption 7 *For every finite field $\mathbb{F}$ and every polynomial $m(k)$ there exists a constant d and a d-variate multi-linear polynomial $P : \mathbb{F}^d \to \mathbb{F}$ such that for every (k, m, d) hypergraph G which is $(t, 2d/3)$-expanding the function $f_{G,P} : \mathbb{F}^k \to \mathbb{F}^m$ is $\exp(\Omega(t))$-pseudorandom over $\mathbb{F}$.*

The constant $2/3$ is somewhat arbitrary and a smaller constant may suffice. (A lower-bound of $1/d$ can be established.) In the binary setting, security was reduced to one-wayness assumption [3] and was analytically established for a large family of algorithms including myopic algorithms, linear tests, statistical algorithms, semi-definite programs and algebraic attacks [6, 7, 24, 48, 11, 39]. Some of these results can be extended to the arithmetic setting as well.

In particular, in [56] it is shown that security against myopic algorithms, linear tests, and low-degree annihilating polynomials holds when the polynomial P is taken to be the *sum-product polynomial* that, for parameters a, b (and arity $d = a + b$), is defined by

$$\mathsf{SP}_{a,b}(w_1, \ldots, w_{a+b}) = (w_1 + \cdots + w_a) + (w_{a+1} \cdot \cdots \cdot w_{a+b}).$$

We note that in the binary case there are linear attacks against this polynomial for quadratic output lengths [11]. Nevertheless, it is shown in [56] that the distinguishing advantage of such attacks is $O(1/|\mathbb{F}|)$, and therefore they do not apply for sufficiently large fields (e.g., whose size is super-polynomial in the security parameter).

On explicit unbalanced constant-degree expanders. In order to employ Assumption 6 and 7 one needs an explicit family of $(k, m = k^{1+\delta}, d = O(1))$ hypergraphs which are $(k^\epsilon, (1 + \Omega(1))d)$-expanding.[18] This assumption is known to be necessary for the existence of d-local (binary) PRG that stretches k bits to m bits [9], and so it was used (either explicitly or implicitly) in previous works that employed such a local PRG (e.g., [35, 40, 42, 41, 2]). See Remark 5.7 in [35] for discussion.

While recent advances in the theory of pseudoranodmness have come close to generating such explicit highly-expanding hypergraphs, in our regime of parameters ($m = \omega(k)$ and $d = O(1)$), an explicit *provable* construction is still unknown. It is important to mention that, by a standard calculation (cf. [46]), a uniformly chosen hypergraph G (i.e., each hyperedge contains a random d-subset of the nodes) is likely to be $(r = \mathrm{poly}(k), 2d/3)$-expanding except with some inverse polynomial failure probability $\epsilon(k)$. Moreover, we can reduce the failure probability to $1/k^c$ for an arbitrary (predetermined) constant c at the expense of increasing the sampling complexity to k^{b_c}, where the constant b grows with c. (This can be done by rejecting hypergraphs which fail to expand for sets of size at most b_c, and re-sampling the hypergraph if needed.) As a result one gets a protocol that fails with "tunable" inverse polynomial probability which is *independent* of the running-time

[18] One can always increase the number of hyperedges to arbitrary polynomial $m = k^a$ at the expense of a minor loss in the other parameters. This can be done by taking a sequence of hypergraphs $G_1, \ldots, G_c$ where G_i is a $(k^{(1+\delta)^{i-1}}, k^{(1+\delta)^i}, d)$-hypergraphs which is (r, bd)-expanding and compose them together (by treating the hyperedges of the i-th graph as the nodes of the $(i+1)$-th hypergraph) and get a $(k, k^{(1+\delta)^i}, D = d^c)$-hypergraph which is $(r/(bd)^{c-1}, bD)$-expanding. Taking c to be a sufficiently large constant (i.e., $\log_{1+\delta} a$), yields the required result.

of the adversary. Moreover, the failure event is restricted to a one-time setup phase and its probability does not increase with the number of times the protocol is executed. Such a guarantee seems to be satisfactory in most practical scenarios. Finally, we mention that there are several heuristic approaches for constructing unbalanced constant-degree expanding hypergraphs. For example, by using some fixed sequence of bits (e.g., the binary expansion of π) and interpreting it as an (k, m, d)-hypergraph via some fixed translation. Assuming such a heuristic to give an explicit construction can be viewed as being a conservative "combinatorial" assumption, in the spirit of standard cryptographic assumptions.

Acknowledgements. The first and fifth authors were supported by the European Union's Horizon 2020 Programme (ERC-StG-2014-2020) under grant agreement no. 639813 ERC-CLC, by an ICRC grant and by the Check Point Institute for Information Security. The third author was supported in part by NSF-BSF grant 2015782, BSF grant 2012366, ISF grant 1709/14, DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1619348, 1228984, 1136174, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the DARPA through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the authors and do not reflect the official policy or position of the DoD, the NSF, or the U.S. Government. The second and forth author were supported by the advanced ERC grant MPCPRO.

References

1. Michael Alekhnovich. More on average case vs approximation complexity. *Computational Complexity*, 20(4):755–786, 2011.
2. Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. *IACR Cryptology ePrint Archive*, 2016:1097, 2016.
3. Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM J. Comput.*, 42(5):2008–2037, 2013.
4. Benny Applebaum. Cryptographic hardness of random local functions - survey. *Computational Complexity*, 25(3):667–722, 2016.
5. Benny Applebaum, Jonathan Avron, and Christina Brzuska. Arithmetic cryptography: Extended abstract. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 143–151, 2015.
6. Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In *STOC*, pages 171–180, 2010.
7. Benny Applebaum, Andrej Bogdanov, and Alon Rosen. A dichotomy for local small-bias generators. In *TCC*, pages 1–18, 2012.
8. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006.
9. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in NC^0. *Computational Complexity*, 17(1):38–69, 2008.
10. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. *SIAM J. Comput.*, 43(2):905–929, 2014.
11. Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In *STOC*, pages 1087–1100, 2016.
12. Sanjeev Arora and Rong Ge. New Algorithms for Learning in Presence of Errors. In *ICALP*, pages 403–415, 2011.
13. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. Cryptology ePrint Archive, Report 2016/602, 2016. `http://eprint.iacr.org/2016/602`.
14. Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, pages 479–488, 1996.
15. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *CRYPTO*, pages 278–291, 1993.
16. Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, 2001.

17. Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.
18. Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 119–136, 2001.
19. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. volume 1992, pages 119–136, 2001.
20. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
21. Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *ITCS*, pages 169–182, 2014.
22. Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies*, volume 5672 of *Lecture Notes in Computer Science*, pages 235–253. Springer, 2009.
23. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *CACM*, 28(6):637–647, 1985.
24. Vitaly Feldman, Will Perkins, and Santosh Vempala. On the complexity of random satisfiability problems with planted solutions. In *STOC*, pages 77–86, 2015.
25. Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 201–210, 2016.
26. E. N. Gilbert. A comparison of signalling alphabets. *Bell System Technical Journal*, 31(3):504–522, 1952.
27. Niv Gilboa. Two party RSA key generation. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 116–129, 1999.
28. O. Goldreich. *Foundations of Cryptography, Voume II Basic Applications*. Cambridge University Press, 2004.
29. Oded Goldreich. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 76–87. 2011.
30. Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudorandom generators. *SIAM J. Comput.*, 22(6):1163–1175, 1993.
31. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.
32. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. volume 2729, pages 145 – 161, 2003.
33. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, 2000.
34. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.
35. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *STOC*, pages 433–442, 2008.
36. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 294–314. Springer, 2009.
37. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure ot extension with optimal overhead. In *Proc. CRYPTO (I)*, pages 724–741, 2015.
38. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM CCS*, pages 830–842, 2016.
39. Pravesh K. Kothari, Ryuhei Mori, Ryan O'Donnell, and David Witmer. Sum of squares lower bounds for refuting any CSP. *CoRR*, abs/1701.04521, 2017.
40. Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In *EUROCRYPT*, pages 28–57, 2016.
41. Huijia Lin. Indistinguishability obfuscation from DDH on 5-linear maps and locality-5 PRGs. *IACR Cryptology ePrint Archive*, 2016:1096, 2016. To appear in *Proc. Crypto 2017*.
42. Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In *FOCS*, pages 11–20, 2016.

43. Michael Luby. LT codes. In *FOCS*, page 271. IEEE Computer Society, 2002.
44. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010*, pages 1–23, 2010.
45. Payman Mohassel and Enav Weinreb. Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries. In *CRYPTO 2008*, pages 481–496, 2008.
46. Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On epsilon-biased generators in NC^0. *Random Struct. Algorithms*, 29(1):56–81, 2006.
47. Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.
48. Ryan O'Donnell and David Witmer. Goldreich's PRG: evidence for near-optimal polynomial stretch. In *CCC*, pages 1–12, 2014.
49. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.
50. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.
51. Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, pages 229–244, 2009.
52. Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Information Theory*, 42(6):1723–1731, 1996.
53. R. R. Varshamov. Estimate of the number of signals in error correcting codes. In *Dokl. Acad. Nauk SSSR*, number 117, page 739–741, 1957.
54. Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Information Theory*, 32(1):54–62, 1986.
55. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.
56. Lior Zichron. Locally computable arithmetic pseudorandom generators. Master's thesis, School of Electrical Engineering, Tel Aviv University, 2017. `http://www.eng.tau.ac.il/~bennyap/pubs/Zichron.pdf`.

A The Rank of Sparse Matrices

In this section we analyze the rank of matrices which are sampled from the distribution $\mathcal{M}(G, \mathbb{F}_p)$ where G is a hypergraph with m hyperedges and k. We begin with the following key observation.

Lemma 5. *Let $\mathbb{F}$ be a field of cardinality $p > 2$ and let G be a hypergraph over k nodes and ℓ hyperedges with the property that every set of nodes S appears in at least $t|S|$ hyperedges for $t = \omega(\log k)$. Then, a random matrix $M \xleftarrow{\$} \mathcal{M}(G, \mathbb{F}_p)$ will have full rank except with probability $\exp(-\Omega(t))$.*

Proof. To prove the claim it suffices to show that

$$\Pr_M[\exists v \neq 0^k \text{ s.t } Mv = 0^\ell] = \exp(-\Omega(t))$$

For a non-empty subset $S \subseteq [k]$, let V_S be the set of all vectors $v \in \mathbb{F}^k$ whose support (set of non-zero coordinates) equals to S. By a union-bound, it suffices to upper-bound

$$\sum_{S \neq \emptyset} q_S, \quad \text{where } q_S = \Pr[\exists v \in V_S \text{ s.t } Mv = 0^\ell]. \tag{3}$$

We will later show that

$$q_S \leq 2^{-|S|(t-1)\log(p-1)} = 2^{-\Omega(|S|t)} \tag{4}$$

Hence we can upper-bound (3) by

$$\sum_{w=1}^{k} \sum_{S:|S|=w} q_S \leq \sum_{w=1}^{k} k^w 2^{-\Omega(wt)} \leq \sum_{w=1}^{k} 2^{-\Omega(wt)} \leq 2^{-\Omega(t)}.$$

It is left to prove (4). Fix a set S of cardinality w, and let us assume without loss of generality that the first t hyperedges of G touch S. Fix some vector $v \in V_S$ and recall that the vector $r_i, i \in [t]$ is sampled by assigning a random non-zero field element to every $j \in [k]$ that participates in the i-th hyperedges. Therefore, every such row is orthogonal to v independently with probability at most $1/(p-1)$. We conclude that, for every $v \in V_S$, we have that

$$\Pr_M[Mv = 0^\ell] \leq (p-1)^{-tw}.$$

By a union-bound, we conclude that

$$q_S \leq \sum_{v \in V_S} \Pr[Mv = 0^\ell] \leq (p-1)^{-w(t-1)},$$

as required. □

Lemma 6. *Let G be a (k, m, d) hypergraph with $m = \omega(kr)$ where $d = O(1)$ and $r = \omega(k \log k)$. Assume that each node of G participates in at least $\Omega(m/k)$ hyperedges. Then, for any field $\mathbb{F}$ of size larger than 2, if we sample $M \xleftarrow{\$} \mathcal{M}(G, \mathbb{F})$ and sub-sample r rows from M, then the resulting matrix M' will have full rank except with negligible probability. Moreover, the above is true even if the rows of M' are sampled from M with replacement.*

For $m = k^3$ and $r = k \log^2 k$, we conclude that the distribution $\mathcal{M}(G, \mathbb{F})$ satisfies the linear-independence condition from Assumption 2.

Proof. Let us describe the sampling procedure in an equivalent way: First sample a hypergraph G' by sub-sampling r hyperedges from G, and then sample M' from $\mathcal{M}(G', \mathbb{F})$. By Lemma 5, it suffices to show that, except with negligible probability, every set S of nodes in G' participates in at least $\omega(\log k)|S|$ hyperedges. Below, we will show that each fixed subset S participates in at least $\omega(\log k)|S|$ hyperedges except with probability $\exp(-\omega(k))$. The theorem therefore follows by a union bound over all 2^k possible subsets.

Fix some non-empty set of nodes S. By assumption, the number of "good" hyperedges in G that touch S is at least $m_0 = |S|\Omega(m/(dk))$. Observe that whenever we sample an hyperedge from M the probability of hitting a good hyperedge is at least $q = (m_0 - r)/m$, regardless of the "history" of the previous samples. (This is true for both sampling with or without replacement.) Therefore, the probability of "failure", i.e., hitting less than $qr/2$ good hyperedges, is upper-bounded by the probability of failure in a binomial experiment where we sample r hyperedges where is good independently with probability q. By a multiplicative Chernoff bound, the probability of seeing less that $qr/2$ successes is at most $\exp(-\Omega(qr))$. Noting that $qr = \Omega(\frac{r|S|}{dk}) - \frac{r^2}{m} = |S|\Omega(r/k) = |S|\omega(\log^2 k)$, concludes the proof. □

By taking G to be the complete $(k, m = \binom{k}{d}, d)$ hypergraph, we derive the following lemma.

Lemma 7. *Let $\mathbb{F}$ be a field of cardinality $p > 2$, and let d be a constant. Then, except with negligible probability in k, a random d-sparse $k \log^2 k \times k$ matrix M over $\mathbb{F}$ has full rank.*

Proof. Let G the complete $(k, m = \binom{k}{d}, d)$ hypergraph and note that the distribution of M can be obtained by sampling $T \xleftarrow{\$} \mathcal{M}(G, \mathbb{F})$ and then sub-sampling $k \log^2 k$ rows from T. The lemma follows from Lemma 6. □