New Protocols for Secure Linear Algebra: Pivoting-Free Elimination and Fast Block-Recursive Matrix Decomposition

Niek J. Bouman[*] Niels de Vreede[†]

Technische Universiteit Eindhoven
the Netherlands
{n.j.bouman,n.d.vreede}@tue.nl

August 22, 2018

Abstract

Cramer and Damgård were the first to propose a constant-rounds protocol for securely solving a linear system of unknown rank over a finite field in multiparty computation (MPC). For m linear equations and n unknowns, and for the case $m \leq n$, the computational complexity of their protocol is $O(n^5)$. Follow-up work (by Cramer, Kiltz, and Padró) proposes another constant-rounds protocol for solving this problem, which has complexity $O(m^4 + n^2m)$. For certain applications, such asymptotic complexities might be prohibitive. In this work, we improve the asymptotic computational complexity of solving a linear system over a finite field, thereby sacrificing the constant-rounds property. We propose two protocols: (1) a protocol based on pivoting-free Gaussian elimination with computational complexity $O(n^3)$ and linear round complexity, and (2) a protocol based on block-recursive matrix decomposition, having $O(n^2)$ computational complexity (assuming "cheap" secure inner products as in Shamir's secret-sharing scheme) and $O(n^{1.585})$ (super-linear) round complexity.

[*]supported by H2020-EU SODA
[†]supported by H2020-EU PRIViLEDGE

1 Introduction

In secure multiparty computation (MPC), n players want to jointly evaluate a function $f(a_1, \ldots, a_n)$ where a_i denotes the input of player i, in such a way that *no player learns anything beyond the output of the function* (and beyond what can be deduced from the output together with that player's input about the other inputs). The latter privacy property should be achieved even in the presence of an adversary that corrupts a subset (of bounded size) of the players. We distinguish between a passive adversary, who merely has access to the views of the corrupted players, and an active adversary, who lets corrupted players deviate in arbitrary ways from the protocol.

MPC protocols come in several flavors and target various scenarios: protocols based on *arithmetic secret-sharing* [BGW88] versus *garbled Boolean circuits* [Yao82] versus *threshold decryption* [CDN01]; *two-party* versus *multi-party* (three or more players) protocols; protocols that are information-theoretically vs. cryptographically secure (by the latter, we mean based on some computational-hardness assumption); protocols that are secure only against passive adversaries, or also against active adversaries (where we can further distinguish between non-adaptive vs. adaptive active adversaries); protocols secure only against an *honest majority* of players versus a *dishonest majority* (in the extreme case, the adversary corrupts all but one player).

In this paper, we revisit the topic of secure linear algebra over finite fields in MPC. We are primarily interested in solving a linear system securely with m linear equations and n unknowns, of unknown rank. We target a multi-party scenario, and we propose protocols that are to be used "on top of" some arithmetic secret-sharing scheme. (For more information about arithmetic-secret-sharing-based MPC, we refer the reader to [CDN15].) By solving a linear system $A\boldsymbol{x} = \boldsymbol{b}$ *securely* for $\boldsymbol{x}$, we mean that given secret-sharings of the elements of the A matrix and the $\boldsymbol{b}$ vector, the players securely compute whether the system has a solution as well as a representation of the space of solutions (a particular solution and A's kernel) in secret-shared form, without revealing any information beyond this (such as the rank). The underlying MPC scheme should be chosen to match the requirements of the targeted scenario, our protocols make black-box use of the underlying scheme via an "arithmetic black box" [DN03], and will inherit the security properties (e.g. security against active vs. passive adversaries) from the underlying scheme.

1.1 Related Work

Cramer and Damgård [CD01] were the first to propose a solution for securely solving a linear system of unknown rank (m linear equations and n unknowns) for the case $m \leq n$, in the form of a constant-rounds protocol, with asymptotic complexity of $O(n^5)$ secure multiplications. Cramer, Kiltz and Padró [CKP07] propose an alternative constant-rounds protocol for this problem based on the Moore–Penrose pseudoinverse. Their protocol has a better asymptotic complexity of $O(m^4 + n^2m)$ secure multiplications.

1.2 This Paper

We propose two new protocols for securely solving the linear system $A\boldsymbol{x} = \boldsymbol{b}$. Our protocols are not constant-rounds but compare favorably in terms of computational complexity.

Our first protocol is based on Gaussian elimination and requires $O(n^3)$ secure multiplications in case A has size $n \times n$. Our second protocol is based on a block-recursive matrix decomposition and has $O(n^\beta)$ complexity, where β is the exponent of the asymptotic complexity of matrix multiplication, where $2 < \beta < 3$ holds for non-oblivious computation. With Shamir's linear secret sharing scheme, we can actually get $\beta = 2$; this is because in Shamir's scheme an inner product between two length-ℓ vectors (for arbitrary ℓ) has the same round complexity and communication complexity as a single secure multiplication.

Gaussian Elimination. Gaussian elimination is arguably the best-known method for solving linear equations. It is well known that Gaussian elimination, without *pivoting*, requires the input matrix A to have a *generic-rank profile* in order to succeed, meaning that the first $r := \operatorname{rank} A$ leading principal minors are nonzero.

Usually pivoting is performed during the course of the algorithm, however, in the context of MPC pivoting is undesirable as it involves row and column swaps, whose positions depend on actual values of the matrix that is being reduced. Performing such operations obliviously is typically very expensive in terms of computational complexity.

In Section 3 we combine a division-free elimination algorithm, as described by Bareiss [Bar68] with a preconditioning method of Kaltofen and Saunders [KS91] to obtain a $O(n^3)$ algorithm with probabilistic correctness.

Fast Block-Recursive Matrix Decomposition. Strassen [Str69] gave the first algorithm for matrix inversion with an asymptotic complexity equal to matrix multiplication. Since Strassen's work, several fast[1] matrix decompositions for obtaining the inverse (or some generalized inverse, in case of a singular matrix) have been proposed in the linear algebra literature. For example, see [JPS13, DPS15] for a recent overview.

An important discriminating characteristic of such algorithms is whether the algorithm's time complexity is *rank sensitive*, i.e., whether the running time depends on the rank of the matrix that is to be decomposed. In many

[1] *Fast* in this context means that the asymptotic computational complexity is the same as that of matrix multiplication.

applications, a rank-sensitive time-complexity is a desired feature because it can significantly reduce the computational work. In *secure* linear algebra we have the opposite situation: because the rank is to remain private, we can only use rank-*in*sensitive algorithms. Another important characteristic of a matrix decomposition algorithm is whether it works for any input matrix, or requires the input to have certain structure, for example, a generic rank-profile.

One example of a decomposition algorithm that works for any input matrix and that has a rank-insensitive complexity is Malaschonok's LEU decomposition [Mal10]. We can then obtain an explicit solution to $A\boldsymbol{x} = \boldsymbol{b}$ in terms of L, E, U and $\boldsymbol{b}$.

2 Preliminaries

Throughout the paper, $\mathbb{K}$ denotes a finite field. For concreteness, you may think of $\mathbb{K}$ as the integers modulo a prime. We use $\mathbb{B} := \{0, 1\} \subset \mathbb{K}$ for the set of bits. We want to emphasize that we define bits as elements of $\mathbb{K}$, such that arithmetic that involves bits and other elements of $\mathbb{K}$ is well-defined.

We write capital letters for matrices, like A, and bold small letters for column vectors, like $\boldsymbol{b}$. We also write $\mathbf{1}$ for the vector $(1, \dots, 1)$, where the length should be clear from its context. If A and B (respectively, $\boldsymbol{b}$) have the same number of rows, then the notation $A|B$ (or $A|\boldsymbol{b}$) means the concatenation of A and B (or A and $\boldsymbol{b}$).

Let $n \in \mathbb{N}$ be nonzero. For any $n \times n$ matrix A with entries in $\mathbb{K}$, we write $\det A$ for the determinant of A. The *leading principal minor of order k* of a matrix B (not necessarily square) is defined as the determinant of the submatrix of B obtained by taking the first k rows and the first k columns of B. Hence, for an $n \times n$ matrix A, the leading principal minor of order n coincides with $\det A$.

Let A be a matrix of rank r. We say that a matrix A has *generic rank profile* [KL96] if for all $k \in [r]$, it holds that A's leading principal minor of order k is nonzero.

Lemma 1 ([KS91, Thm. 2])**.** *Let $A \in \mathbb{K}^{m\times n}$ be arbitrary and let $r := \operatorname{rank} A$. Consider the matrix $A' := UAL$ with*

$$U := \begin{pmatrix} 1 & u_2 & u_3 & \dots & u_m \\ & 1 & u_2 & \dots & u_{m-1} \\ & & 1 & \ddots & \vdots \\ & & & \ddots & u_2 \\ & & & & 1 \end{pmatrix}, \quad \textit{and} \quad L := \begin{pmatrix} 1 & & & & \\ \ell_2 & 1 & & & \\ \ell_3 & \ell_2 & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_n & \ell_{n-1} & \dots & \ell_2 & 1 \end{pmatrix},$$

where the elements of the unit upper triangular Toeplitz matrix U and the elements of the unit lower triangular Toeplitz matrix L are selected independently and uniformly at random from $\mathbb{K}$. Then, the probability that A' has the *property that the leading principal minor of order k is nonzero for all $k \in [r]$ is bounded below by*

$$\Pr(A' \textit{ has generic rank profile }) \geq 1 - \frac{r(r+1)}{|\mathbb{K}|}.$$

We say that *preconditioning fails* in the event that A' does not have generic rank profile.

For a vector $(v_1, \ldots, v_n)$, we write $\operatorname{diag}(v_1, \ldots, v_n)$ for the $n \times n$ diagonal matrix with $v_1, \ldots, v_n$ as its diagonal elements. We denote the $m \times n$ matrix that has ones on the diagonal and is zero elsewhere as $I_{m\times n}$. Multiplying a matrix by $I_{m\times n}$ corresponds to adding or removing rows or columns. The $n \times n$ identity matrix is denoted I_n.

For any $a \in \mathbb{K}$, we write $[\![a]\!]$ for the secret-shared version of a. As mentioned in the introduction, we assume the availability of an arithmetic black-box, by which we mean that there is some underlying MPC scheme that provides, for any $a, b, c \in \mathbb{K}$, $[\![a]\!] + [\![b]\!]$ (addition), $[\![a]\!] \cdot [\![b]\!]$ (multiplication), $[\![a]\!] + c$ (addition by a public constant), $[\![a]\!] \cdot c$ (multiplication by a public constant), and $[\![a]\!]^{-1}$ (multiplicative inverse), as well as an operation that allows a player to create a secret-sharing from a privately held value, and an operation to reveal a secret-shared value to a designated subset of the players. By the notation $[\![a]\!] \overset{?}{=} 0$, we mean that an MPC protocol is invoked to perform a secure equality test (see, e.g., [NO07]), which returns $[\![1]\!]$ if the logical assertion is true and $[\![0]\!]$ otherwise. For a matrix $A \in \mathbb{K}^{m\times n}$, $[\![A]\!]$ denotes the matrix that is element-wise secret-shared. Operations like matrix transpose and matrix multiplication are naturally defined on such secret-shared matrices.

3 Division-Free and Pivoting-Free Gaussian Elimination

In this section we present our protocol for secure pivoting-free Gaussian elimination. The ObliviousGE protocol, as displayed in Protocol 1, is based on Bareiss' division-free algorithm for integer-preserving Gaussian elimination [Bar68] and uses the preconditioning method, Lemma 1, by Kaltofen and Saunders [KS91] to avoid the need for pivoting. The protocol is suitable for solving overdetermined, as well as underdetermined systems of unknown rank, and can be used to simultaneously solve $A\boldsymbol{x} = \boldsymbol{b}$ for multiple $\boldsymbol{b}$ vectors, by arranging these vectors in a matrix B.

The system $A\boldsymbol{x} = \boldsymbol{b}$ is solvable if and only if $\operatorname{rank} A = \operatorname{rank}(A \mid \boldsymbol{b})$. If the system is underdetermined yet solvable, then the space of solutions is given by $\{\boldsymbol{x} + \boldsymbol{q} \mid \boldsymbol{q} \in \operatorname{Ker} A\}$ where $\boldsymbol{x}$ is a particular solution. To represent all possible solutions, our protocol also obliviously computes a basis for $\operatorname{Ker} A$.

The ObliviousGE protocol takes as input a secret-shared pair of matrices $A \in \mathbb{K}^{m\times n}$ and $B \in \mathbb{K}^{m\times \ell}$, where each column $\boldsymbol{b}$ of B represents a vector for which we wish to solve $A\boldsymbol{x} = \boldsymbol{b}$. Let $\mu := \min(m, n)$ and $r := \operatorname{rank} A$, then $r \leq \mu$. The output of our protocol consists of secret shares of:

$\boldsymbol{s} \in \mathbb{B}^{\ell}$,	a Boolean vector indicating for which columns of B a solution exists;
$X \in \mathbb{K}^{n \times \ell}$,	whose columns contain the solution to the corresponding system, if it exists;
$Q \in \mathbb{K}^{n \times n}$,	whose r rightmost columns form a basis for $\operatorname{Ker} A$;
$\boldsymbol{r} \in \mathbb{B}^{\mu}$,	the unary representation of $\operatorname{rank} A$; and
$d \in \mathbb{K}$,	the determinant of A, if A is square.

Because our protocol is oblivious, the dimensions of X and Q must not depend on the rank of A; they are determined solely by the dimensions of A and B. The columns of X corresponding to inconsistent systems and the superfluous columns of Q are therefore set to zero.

Protocol 1 invokes SampleToeplitzPrecond as a subprotocol to sample random upper- and lower-unitriangular Toeplitz preconditioner matrices as described in Lemma 1.

Protocol 1 also invokes RandomVector as a subprotocol (with parameter n), which returns a vector of n public elements, each sampled independently and uniformly at random from the multiplicative group $\mathbb{K}^*$.

In a naïve adaptation of Bareiss' division-free algorithm to the unknown-rank case, one would additionally perform division by $c_{k-1,k-1}$ on line 13 of Protocol 1 in each iteration except the first. Instead, we postpone those division operations to the end of the protocol, where they can be amortized as an element-wise multiplication by the inverse of a row-dependent divisor.

Let U and L denote, respectively, the upper and lower triangular Toeplitz preconditioner matrix. To compute X and Q simultaneously, our algorithm performs elimination on the matrix

$$C := \begin{pmatrix} A' & B' \\ I_n & 0 \end{pmatrix} = \begin{pmatrix} UAL & UB \\ I_n & 0 \end{pmatrix},$$

where A' and B' denote the preconditioned versions of A and B respectively. The lower-right zero block matrix in C is actually irrelevant, hence our algorithm always skips those entries. During elimination, (scaled) particular solutions to $A'X_i = B'_i$ for all $i \in [m]$ for which $s_i = 1$, as well as a basis for $\operatorname{Ker} A'$, appear at the B'-part and I_n-part of C, respectively, in the r-th iteration, where r is the rank of A. At this point, the remaining rows (i.e., those below the r-th row) in the A-part and in the columns of the B-part that correspond to consistent subsystems are all zero. For the sake of data independence, the algorithm must continue performing elimination steps after the r-th iteration. Therefore, the remaining rows are obliviously replaced by the corresponding rows from the identity matrix, ensuring that the remaining elimination operations do not further modify the results.

Protocol 1 $(\llbracket \boldsymbol{s} \rrbracket, \llbracket X \rrbracket, \llbracket Q \rrbracket, \llbracket \boldsymbol{r} \rrbracket, \llbracket d \rrbracket) \leftarrow \mathsf{ObliviousGE}(\llbracket A \rrbracket, \llbracket B \rrbracket)$

```
Input:  $A \in \mathbb{K}^{m\times n}$, $B \in \mathbb{K}^{m\times \ell}$
Output:  $\boldsymbol{s} \in \mathbb{B}^{\ell}$, $X \in \mathbb{K}^{n\times \ell}$, $Q \in \mathbb{K}^{n\times n}$, $\boldsymbol{r} \in \mathbb{B}^{\mu}$, $d \in \mathbb{K}$, where $\mu = \min(m, n)$
1: $(U, L) \leftarrow \mathsf{SampleToeplitzPrecond}(m, n)$
2: $\boldsymbol{z} \leftarrow \mathsf{RandomVector}(m)$
3: $\llbracket C \rrbracket \leftarrow \begin{pmatrix} I_{\mu\times m}U\llbracket A \rrbracket L & I_{\mu\times m}U\llbracket B \rrbracket \\ \llbracket I_n \rrbracket & 0^{n\times\ell} \end{pmatrix}$,   $\llbracket h \rrbracket \leftarrow \llbracket 1 \rrbracket$,   $\llbracket t \rrbracket \leftarrow \llbracket 1 \rrbracket$
4: for $k \leftarrow 1$ to $\mu$ do
5:     $\llbracket r_k \rrbracket \leftarrow \llbracket c_{k,k} \rrbracket \overset{?}{\neq} 0$
6:     $\llbracket c_{\mu+k,k} \rrbracket \leftarrow \llbracket h \rrbracket$
7:     $\llbracket f_k \rrbracket \leftarrow \llbracket h \rrbracket$
8:     $\llbracket t \rrbracket \leftarrow \llbracket t \rrbracket \cdot \llbracket h \rrbracket$
9:     $\llbracket h \rrbracket \leftarrow \llbracket h \rrbracket \cdot (\llbracket c_{k,k} \rrbracket + 1 - \llbracket r_k \rrbracket)$
10:    for $i \leftarrow 1$ to $\mu + k$ do
11:        for $j \leftarrow k + 1$ to $n + \ell$ do
12:            if $i \neq k \wedge (i \leq \mu \vee j \leq n)$ then
13:                $\llbracket c_{i,j} \rrbracket \leftarrow \big((\llbracket c_{k,k} \rrbracket + 1 - \llbracket r_k \rrbracket) \quad \llbracket c_{k,j} \rrbracket\big) \cdot \begin{pmatrix} \llbracket c_{i,j} \rrbracket \\ -\llbracket c_{i,k} \rrbracket \end{pmatrix}$
14: for $k \leftarrow \mu + 1$ to $n$ do
15:     $\llbracket c_{\mu+k,\mu+k} \rrbracket \leftarrow \llbracket h \rrbracket$
16: $\begin{pmatrix} & \llbracket X \rrbracket \\ \llbracket Q \rrbracket & \end{pmatrix} \leftarrow C$          ▷ $Q \in \mathbb{K}^{n\times n}$ and $X \in \mathbb{K}^{\mu\times\ell}$
17: $\llbracket Q \rrbracket \leftarrow L\llbracket Q \rrbracket \operatorname{diag}(\mathbf{1} - \llbracket \boldsymbol{r} \rrbracket)$
18: $\llbracket g \rrbracket \leftarrow (\llbracket t \rrbracket \cdot \llbracket h \rrbracket)^{-1}$
19: $\llbracket X \rrbracket \leftarrow \llbracket g \rrbracket \cdot \llbracket t \rrbracket L I_{n\times\mu} \operatorname{diag}(\llbracket \boldsymbol{f} \rrbracket)\llbracket X \rrbracket$
20: $\llbracket \boldsymbol{s} \rrbracket \leftarrow \boldsymbol{z}\llbracket A \rrbracket\llbracket X \rrbracket - \boldsymbol{z}\llbracket B \rrbracket$
21: for $j \leftarrow 1$ to $\ell$ do
22:     $\llbracket s_j \rrbracket \leftarrow \llbracket s_j \rrbracket \overset{?}{=} 0$
23: $\llbracket X \rrbracket \leftarrow \llbracket X \rrbracket \operatorname{diag}(\llbracket \boldsymbol{s} \rrbracket)$
24: if $m = n$ then $\llbracket d \rrbracket \leftarrow \llbracket r_\mu \rrbracket \cdot \llbracket g \rrbracket \cdot \llbracket h \rrbracket \cdot \llbracket h \rrbracket$
25: else $\llbracket d \rrbracket \leftarrow \llbracket 0 \rrbracket$
26: return $\llbracket \boldsymbol{s} \rrbracket, \llbracket X \rrbracket, \llbracket Q \rrbracket, \llbracket \boldsymbol{r} \rrbracket, \llbracket d \rrbracket$
```

3.1 Analysis

It will be helpful to view Protocol 1 as the composition of two parts: a *sampling part* (lines 1 and 2) where uniformly random field elements are sampled independently of the algorithm's input, and a *circuit part* (from line 3 onwards).

3.1.1 Complexity Analysis

Computational Complexity. Let M_{inv}, M_{rand} and M_{zt} be the complexity of securely performing a inversion of a field element, uniformly random sampling of a public field element and a zero test, respectively, expressed terms of the complexity of a secure multiplication. The complexity of these operations is independent of the dimensions of the system. The computational complexity of the k-th iteration of the main loop is then equal to $(\mu+k-1)(n+\ell-k)-n\ell+2+M_{\mathrm{zt}}$. Summing this over μ iterations and including the $n(n+\ell)+(\mu+1)(\ell+1)+4+(2m+n-2)M_{\mathrm{rand}}+\ell M_{\mathrm{zt}}$ operations outside the main loop, gives a computational complexity of $\frac{1}{6}\mu^2(9n-5\mu)+\frac{1}{2}\ell n^2+O(n^2+n\ell+m)$.

If $m \geq n$, but $m = o(n^3)$ and $\ell = o(n)$, the computational complexity simplifies to $\frac{2}{3}n^3+o(n^3)$.

Round Complexity. Let R_{mul}, R_{inv}, R_{rand} and R_{zt} be the number of rounds needed to securely perform a multiplication, inversion of a field element, uniformly random sampling of a public field element and a zero test, respectively. Protocol 1 then runs in $(\mu+4)R_{\mathrm{mul}}+R_{\mathrm{inv}}+R_{\mathrm{rand}}+(\mu+1)R_{\mathrm{zt}}$ rounds.

Because constant rounds protocols are known for these four operations, the round complexity of Protocol 1 is $O(\mu)=O(\min(m,n))$.

3.1.2 Privacy

As for the privacy of our algorithm, we have the following claim.

Proposition 2. *Let ε_{zt} denote the error-probability of the secure zero test used in Algorithm 1. When applied to inputs $A \in \mathbb{K}^{m\times n}$ and $B \in \mathbb{K}^{m\times \ell}$, Algorithm 1 reveals no private information except with probability at most*

$$\varepsilon \leq \frac{\mu(\mu+1)+\ell}{|\mathbb{K}|-1}+(\mu+\ell)\varepsilon_{\mathrm{zt}}.$$

Proof. From Lemma 1, we know that preconditioning fails to turn A into a matrix with generic rank profile with probability less than $r(r+1)/p$, where $r := \operatorname{rank} A$ and $p := |\mathbb{K}|$ is the modulus of the MPC scheme. Because r is unknown, we substitute μ as tight upper bound for r.

On lines 20 to 22 we test, for all $j \in [\ell]$, whether the vector consisting of the j-th column of $AX-B$ equals the zero-vector by taking an inner product with the (public) vector $\boldsymbol{z}$ of field elements uniformly and independently sampled from the multiplicative group $\mathbb{K}^*$, and applying a secure zero test to this inner product. The probability of the event that the inner product vanishes while the column is not identical to the zero vector, the failure probability, is $1/(p-1)$. Because we perform this zero-vector-test ℓ times, we apply the union bound and conclude that the failure probability over ℓ applications is $\ell/(p-1)$.

Note that in total we perform $\mu+\ell$ secure zero tests on lines 5 and 22, which gives us (again by a union-bound argument) an additive term of $(\mu+\ell)\varepsilon_{\mathrm{zt}}$.

Apart from the zero tests, the only non-elementary operation performed in the circuit part of Protocol 1 is the field inversion on line 18. Because h and t are initialized to 1 and are only ever modified through multiplication by values from the multiplicative group $\mathbb{K}^*$, unless one of the zero tests on line 5 fails, their product on line 18 is also an element of $\mathbb{K}^*$ and the inverse can be computed using, e.g., the field inversion protocol of [BIB89] without revealing any information.

Privacy of the circuit part of Protocols 1 then follows immediately from the privacy properties of the underlying MPC scheme. □

3.1.3 Correctness

To prove correctness, we take Algorithm 1 as starting point, which we assume to be correct with probability $1-\eta$, where η represents the failure probability of the Toeplitz preconditioner, and transform this algorithm in several steps into Protocol 1. It then suffices to prove that each transformational step preserves correctness.

Base algorithm. Algorithm 1 is a Gaussian elimination algorithm for solving a linear system of arbitrary rank. The algorithm takes as input pair of matrices $A \in \mathbb{K}^{m \times n}$ and $B \in \mathbb{K}^{m \times \ell}$ and an integer $r = \operatorname{rank}(A)$. The algorithm computes a pair $\boldsymbol{s} \in \mathbb{B}^{\ell}$ and $X \in \mathbb{K}^{n \times \ell}$ such that $AX \operatorname{diag}(\boldsymbol{s}) = B \operatorname{diag}(\boldsymbol{s})$ and $\boldsymbol{s}$ is maximal. Furthermore, the algorithm produces $Q \in \mathbb{K}^{n \times n}$, a basis for the nullspace of A and $d \in \mathbb{K}$, which equals $\det(A)$ if A is square and should be ignored otherwise.

Algorithm 1 uses Toeplitz preconditioning [KS91] and ℓ probabilistic zero-vector tests. The preconditioning ensures that, except with probability η, elimination can be performed without pivoting. The algorithm fails if preconditioning fails, or any of the probabilistic zero tests give false positives.

Since Algorithm 1 forms the basis for our protocol for oblivious Gaussian elimination, the size of the output of the algorithm may only depend on the size of the input. This is why the algorithm must return even those columns of X that correspond to inconsistent columns of B. Similarly, the dimension of the nullspace is $n-r$, but in the final protocol, r remains unknown, so we represent the basis of the nullspace in A by r zero columns, followed by $n-r$ columns which span the nullspace.

After sampling the randomness needed for the probabilistic zero-vector test and the preconditioning, Algorithm 1 augments the input matrix A to the horizontally with B and vertically with I_n, filling the remaining positions with zeros and immediately applies the preconditioners forming the matrix $C \in \mathbb{K}^{(m+n)\times(n+\ell)}$. The algorithm then performs Gaussian elimination by applying elementary row operations until the A part of C is in row echelon form. It then extracts the basis for the nullspace Q and partial solution X' to the preconditioned problem. X' is then resized to the correct dimensions and the candidate solution to the non-preconditioned problem is extracted. Each of the columns of the candidate solution is tested and finally, the value of d is computed.

Algorithm 1 Base algorithm for Gaussian elimination

```
Input: $A \in \mathbb{K}^{m\times n}$, $B \in \mathbb{K}^{m\times \ell}$, $r \in \mathbb{N}$
Output: $\boldsymbol{s} \in \mathbb{B}^{\ell}$, $X \in \mathbb{K}^{n\times \ell}$, $Q \in \mathbb{K}^{n\times n}$, $d \in \mathbb{K}$
 1: $\boldsymbol{z} \leftarrow \mathsf{RandomVector}(m)$
 2: $(U, L) \leftarrow \mathsf{SampleToeplitzPrecond}(m, n)$
 3: $C^{(0)} \leftarrow \begin{pmatrix} UAL & UB \\ I & 0^{n\times\ell} \end{pmatrix}$
 4: for $k \leftarrow 1$ to $r$ do
 5:     for $j \leftarrow 1$ to $n+\ell$ do
 6:         $c^{(k)}_{k,j} \leftarrow c^{(k-1)}_{k,j} \cdot (c^{(k-1)}_{k,k})^{-1}$
 7:     for $i \leftarrow 1$ to $m+n$ except $k$ do
 8:         for $j \leftarrow 1$ to $n+\ell$ do
 9:             $c^{(k)}_{i,j} \leftarrow c^{(k-1)}_{i,j} - c^{(k-1)}_{i,k} \cdot c^{(k)}_{k,j}$
10: $\begin{pmatrix} & X' \\ Q' & \end{pmatrix} \leftarrow C^{(k)}$          ▷ $Q \in \mathbb{K}^{n\times n}$ and $X \in \mathbb{K}^{m\times \ell}$
11: $Q \leftarrow LQ'$
12: $X \leftarrow LI_{n\times m}X'$
13: $\boldsymbol{v} \leftarrow \boldsymbol{z}AX - \boldsymbol{z}B$
14: for $j \leftarrow 1$ to $\ell$ do
15:     $s_j \leftarrow v_j \overset{?}{=} 0$
16: $d \leftarrow \prod_{k=1}^{r} c^{(k-1)}_{k,k}$
17: return $\boldsymbol{s}, X, Q, d$
```

Rank independence. We will now transform the base algorithm into a rank-independent form. This means that the algorithm will not take the rank as input and does not branch depending on the rank. The rank-independence transformation of the base algorithm is displayed in Algorithm 2. We will use the fact that the rank cannot exceed $\mu = \min(m, n)$.

Since the upper bound of the main loop in the base algorithm (which runs over k) depends on the rank, we modify this loop such that it runs over the first μ rows of C (line 4). This this modification may increase the number of iterations, hence we must ensure that any additional iterations do not affect the result compared to the base algorithm and we must do so without rank-dependent branching. We achieve this by testing whether the pivot element is zero on line 5 and make incorporate the test result into the computation described on line 7.

To show that the operations on lines 7 and 10 leave the state of the matrix unchanged in iterations $k > r$, first observe that, unless preconditioning failed,

$c_{k,k}^{(k-1)} = 0$ in iteration k if and only if $k > r$. Furthermore, in this case $c_{k,j}^{(k-1)} = 0$ for all j except those corresponding to inconsistent columns in the B part. Therefore, if $k \leq r$, we have that $r_k = 1$ and line 7 performs the same operations as in the base algorithm. If $k > r$, we have that $r_k = 0$ and $c_{k,j}^{(k-1)} = 0$ for all j, except for inconsistent columns. Substituting these values shows that line 7 reduces to $c_{k,j}^{(k)} \leftarrow c_{k,j}^{(k-1)}$ and, with the exception of inconsistent columns, line 10 reduces to $c_{i,j}^{(k)} \leftarrow c_{i,j}^{(k-1)}$, i.e., these steps do not perform any computation. We therefore conclude that, with the exception of inconsistent columns, $C^{(\mu)} = C^{(r)}$ in the rank-independent algorithm and that these are equal to $C^{(r)}$ in the base algorithm.

The fact that the inconsistent columns of $C^{(\mu)}$ in the rank-independent algorithm do not necessarily agree with the inconsistent columns of $C^{(r)}$ in the base algorithm does not pose any problems, as each of these columns is subjected to the correctness test. Since no solution to $A\boldsymbol{x} = \boldsymbol{b}$ exists for inconsistent $\boldsymbol{b}$, any such deviation from the base algorithm cannot affect the error probability of the correctness test.

Finally, the output of the rank-independent algorithm includes the unary encoding of rank(A).

Delayed Division. Algorithm 2 performs a field inversion in every iteration. If we were to transform Algorithm 2 into a secure protocol as is, this would require invoking a secure field-inversion protocol in every iteration and, since the result is immediately used in the same iteration, it would not be possible to schedule the execution of the field-inversion operation in parallel with the remaining elimination operations from that iteration. Hence, we reduce both the number of secure operations and the round complexity of the oblivious Gaussian elimination protocol by accumulating the divisors and performing the division after the main loop.

Let $\tilde{C}^{(k)}$ denote the state of the matrix after the k-th iteration in the algorithm modified to delay the division, with $\tilde{C}^{(0)} = C^{(0)}$. In the modified algorithm $\tilde{C}^{(k)}$ is computed by omitting line 7 from Algorithm 2. Then $\tilde{c}_{k,j}^{(k)} = \tilde{c}_{k,j}^{(k-1)}$ for all j. If we were to use these values in the computation on line 10, then the second term would be scaled up by some factor, while the first would remain the same. To keep the result consistent, we must also scale the first term up by the same factor. This can be achieved by replacing line 10 with

$$\tilde{c}_{i,j}^{(k)} \leftarrow (\tilde{c}_{k,k}^{(k-1)} + 1 - r_k) \cdot \tilde{c}_{i,j}^{(k-1)} - \tilde{c}_{i,k}^{(k-1)} \cdot \tilde{c}_{k,j}^{(k-1)}. \qquad (1)$$

In Protocol 1 this operation is actually expressed as a dot product on line 13 to emphasize that we can exploit the efficient dot-product protocol to reduce the number of multiplications needed by a factor of two.

The above modifications keep the matrix consistent, but to compute the actual result, we also need to determine the factors by which we have to

Algorithm 2 Rank-independent Gaussian elimination

```
Input:  $A \in \mathbb{K}^{m\times n}$, $B \in \mathbb{K}^{m\times \ell}$
Output:  $\boldsymbol{s} \in \mathbb{B}^{\ell}$, $X \in \mathbb{K}^{n\times \ell}$, $Q \in \mathbb{K}^{n\times n}$, $\boldsymbol{r} \in \mathbb{B}^{\mu}$, $d \in \mathbb{K}$, where $\mu = \min(m,n)$
 1: $\boldsymbol{z} \leftarrow \mathsf{RandomVector}(m)$
 2: $(U, L) \leftarrow \mathsf{SampleToeplitzPrecond}(m, n)$
 3: $C^{(0)} \leftarrow \begin{pmatrix} UAL & UB \\ I & 0^{n\times\ell} \end{pmatrix}$
 4: for $k \leftarrow 1$ to $\mu$ do
 5:     $r_k \leftarrow c^{(k-1)}_{k,k} \overset{?}{\neq} 0$
 6:     for $j \leftarrow 1$ to $n+\ell$ do
 7:         $c^{(k)}_{k,j} \leftarrow c^{(k-1)}_{k,j} \cdot (c^{(k-1)}_{k,k} + 1 - r_k)^{-1}$
 8:     for $i \leftarrow 1$ to $m+n$ except $k$ do
 9:         for $j \leftarrow 1$ to $n+\ell$ do
10:             $c^{(k)}_{i,j} \leftarrow c^{(k-1)}_{i,j} - c^{(k-1)}_{i,k} \cdot c^{(k)}_{k,j}$
11: $\begin{pmatrix} & X' \\ Q' & \end{pmatrix} \leftarrow C^{(k)}$        ▷ $Q \in \mathbb{K}^{n\times n}$ and $X \in \mathbb{K}^{m\times \ell}$
12: $Q \leftarrow LQ'$
13: $X \leftarrow LI_{n\times m}X'$
14: $\boldsymbol{v} \leftarrow \boldsymbol{z}AX - \boldsymbol{z}B$
15: for $j \leftarrow 1$ to $\ell$ do
16:     $s_j \leftarrow v_j \overset{?}{=} 0$
17: $d \leftarrow \prod_{k=1}^{\mu} c^{(k-1)}_{k,k}$
18: return $\boldsymbol{s}, X, Q, \boldsymbol{r}, d$
```

compensate the outcome of the main loop. For every k, let $\hat{F}^{(k)}$ be the diagonal matrix containing the ratios of the rows of $\tilde{C}^{(k)}$ and the rows of $C^{(k)}$, i.e., $\tilde{C}^{(k)} = \hat{F}^{(k)}C^{(k)}$, with $\hat{F}^{(0)} = I$. First, we have that

$$\hat{f}^{(k)}_k = \tilde{c}^{(k)}_{k,k} + 1 - r_k$$

satisfies $\tilde{C}^{(k)} = \hat{F}^{(k)}C^{(k)}$, since $c^{(k)}_{k,k} = r_k$. Next, for all $i \neq k$, careful inspection of (1) shows that

$$\hat{f}^{(k)}_i = \hat{f}^{(k-1)}_i \hat{f}^{(k)}_k = \hat{f}^{(k-1)}_i (\tilde{c}^{(k)}_{k,k} + 1 - r_k).$$

From this we conclude for all i that

$$\hat{f}^{(k)}_i = \begin{cases} \prod_{j=i}^{k} (\tilde{c}^{(j)}_{j,j} + 1 - r_j) = \hat{f}^{(k)}_1 (\hat{f}^{(i-1)}_1)^{-1} & \text{if } i \leq k \\ \prod_{j=1}^{k} (\tilde{c}^{(j)}_{j,j} + 1 - r_j) = \hat{f}^{(k)}_1 & \text{otherwise.} \end{cases}$$

This form allows us to reconstruct $C^{(\mu)} = (\hat{F}^{(\mu)})^{-1}\tilde{C}^{(\mu)}$ using a single field inversion as

$$(\hat{F}^{(\mu)})^{-1} = (\hat{f}^{(\mu)}_1)^{-1} \operatorname{diag}\begin{pmatrix} 1 & \hat{f}^{(1)}_1 & \hat{f}^{(2)}_1 & \ldots & \hat{f}^{(\mu-1)}_1 & 1 & \ldots & 1 \end{pmatrix}.$$

In addition to reconstructing $C^{(\mu)}$, we must also reconstruct d, the determinant (if it exists). On line 17 of Algorithm 2, the d is computed as $\prod_{k=1}^{\mu} c_{k,k}^{(k-1)}$. The determinant exists if and only if $m = n = \mu$ and is non-zero if and only if $\boldsymbol{r}$ is the all-ones vector, which is equivalent to $r_\mu = 1$ under the assumption that no preconditioning error occurred. We can then compute the determinant as

$$\begin{aligned} d = r_\mu \prod_{k=1}^{\mu} c_{k,k}^{(k-1)} &= r_\mu \prod_{k=1}^{\mu} (\hat{f}_k^{(k-1)})^{-1} \tilde{c}_{k,k}^{(k-1)} \\ &= r_\mu \left(\prod_{k=1}^{\mu} \hat{f}_k^{(k-1)} \right)^{-1} \prod_{k=1}^{\mu} (\tilde{c}_{k,k}^{(k)} + 1 - r_k) = r_\mu \hat{f}_1^{(\mu)} \left(\prod_{k=1}^{\mu-1} \hat{f}_1^{(k)} \right)^{-1} . \end{aligned} \tag{2}$$

Finally, we compute the two field inversions that we need in Equations (3.1.3) and (2) from $\left(\prod_{k=1}^{\mu} \hat{f}_1^{(k)}\right)^{-1}$ as

$$\begin{aligned} (\hat{f}_1^{(\mu)})^{-1} &= \prod_{k=1}^{\mu-1} \hat{f}_1^{(k)} \left(\prod_{k=1}^{\mu} \hat{f}_1^{(k)} \right)^{-1} ; \text{ and} \\ \left(\prod_{k=1}^{\mu-1} \hat{f}_1^{(k)} \right)^{-1} &= \hat{f}_1^{(\mu)} \left(\prod_{k=1}^{\mu} \hat{f}_1^{(k)} \right)^{-1} . \end{aligned}$$

It is straightforward to see that at the end of the k-th iteration of the main loop in Protocol 1, $h = \hat{f}_1^{(k)}$, $f_k = \hat{f}_1^{(k-1)}$, and $t = \prod_{j=1}^{k-1} \hat{f}_1^{(j)}$. Hence, the following equation holds for g (defined on line 18): $g = \left(\prod_{k=1}^{\mu} \hat{f}_1^{(k)}\right)^{-1}$, and the use of g to normalize the candidate solution (line 19) and to compute the determinant (line 24) is consistent with the derivation given above.

Omitting Unnecessary Multiplications. On line 11 of Algorithm 2 only the top-right and bottom-left block of the matrix are extracted; the other blocks are no longer used in the computation. This suggests that we can omit computing some of these values in the main loop. Indeed, no value from the bottom right block is ever used to determine the output. In Protocol 1, we do not perform any computation on this block. Similarly, the values in the k-th column of the top left block are no longer used after the k-th iteration and computations on these columns are also omitted.

We know that the values in rows $\mu + 1$ through m of C, if any, never affect the values of any of the other rows and that these rows are zero after the main loop, except for inconsistent columns in the B-part. We explicitly ignore these rows by multiplying the input matrices by $I_{\mu \times m}$ from the left after preconditioning.

It is also possible to reduce the number of multiplications needed to perform one iteration on the bottom left block. It holds that after the k-th iteration, where $k \leq r$, the first k columns of the bottom left block of C are zero. Furthermore, only the first k rows of this block are filled with arbitrary values. The remaining rows only have a non-zero element on the diagonal. In particular, the value of these diagonal elements after the k-th iteration is equal $\prod_{j=1}^{k-1}(\tilde{c}_{j,j}^{(j)} + 1 - r_k)$, which is equal to h *before* the k-th iteration. To see that this holds, first consider that these properties hold true before the first iteration, as the bottom left block of C is initialized to the $n \times n$ identity matrix. If these properties hold after the k-th iteration, with $k < r$, then after the $k+1$-st iteration, the first $k+1$ rows are updated to contain zeros in the $k+1$-st column by the ordinary Gaussian elimination process. This update does not introduce any non-zero values in the preceding columns. For the remaining rows, $c_{i,k} = 0$ and $c_{i,j} = 0$ for $j \neq i$, i.e., on the off-diagonal elements, meaning that the result of the Gaussian elimination step would keep these elements equal to zero. The diagonal elements, meanwhile, would be multiplied by $\tilde{c}_{k,k}^{(k)} + 1 - r_k$, which results in the stated property. We can therefore omit computing the first k columns of the *entire* matrix in the k-th iteration and similarly, omit computation beyond the first k rows of the bottom left block as long as we keep track of the value that the diagonal elements should assume as soon as they are used.

The omission of these unnecessary Gaussian elimination steps is reflected on lines 6, 10–12, 14 and 15 of Protocol 1, with lines 10–12 selecting only those elements on which Gaussian elimination should be performed and lines 6, 14 and 15 placing the correct values on the diagonal of the bottom left block of C when needed.

As in Algorithm 2, the modified Gaussian elimination step that is performed when $k > r$ actually leaves C unchanged, except for inconsistent columns. Since h also remains unchanged in this case, the operations on lines 6, 14 and 15 insert the correct value on the diagonal of the bottom left block.

Because the protocol does not actually complete the Gaussian elimination process on the first r columns of the bottom left block, we must set those columns to zero after the main loop. This is performed obliviously on line 17

Explicit Zeros. For the final modification, we explicitly set the columns of X corresponding to inconsistent systems to zero on line 23. Additionally, we set d to zero on line 25, in case the matrix is not square and therefore does not have a determinant. This modification is not strictly necessary for correctness, but omitting it would leave part of the output undefined, which may be considered bad practice for a secure protocol.

4 Solving $Ax = b$ via Block-Recursive Matrix Decomposition

In [Mal10], Malaschonok proposes and proves the correctness of a recursive algorithm for the *LEU* decomposition, with a rank-insensitive time complexity.

Let $\mathbb{K}$ be a finite field. Let $\mathcal{L}_n, \mathcal{U}_n \subset \mathbb{K}^{n\times n}$ for all $n \in \mathbb{N}$ denote respectively the subalgebra of lower and upper n-by-n triangular matrices over $\mathbb{K}$. Let $\mathcal{P}_n$ for any $n \in \mathbb{N}$ denote the set of matrices in $\mathbb{B}^{n\times n}$ whose rank equals the number of non-zero elements.

For any matrix $A \in \mathbb{K}^{2^d\times 2^d}$ for any $d \in \mathbb{N}$ of arbitrary rank $r \in \{0, \dots, d\}$, the *LEU* decomposition is defined as

$$LAU = E,$$

where $L \in \mathcal{L}_{2^d}$ is invertible, $E \in \mathcal{P}_{2^d}$ of rank r, and $U \in \mathcal{U}_{2^d}$ is unitriangular (hence invertible).

The structure of the *LEU* decomposition implies the following.

Proposition 3. *Let $n = 2^d$ for $d \in \mathbb{N}$. For any square matrix $A \in \mathbb{K}^{n\times n}$, let (L, E, U) form the LEU decomposition of A. Let $s := n - \operatorname{rank} A$. Then,*

(i) the s columns in U, for which the corresponding s columns in E are equal to the zero vector, form a basis for $\operatorname{Ker}(A)$*;*

(ii) the s rows in L, for which the corresponding s rows in E are equal to the zero vector, form a basis for $\operatorname{Ker}(A^\intercal)$.

Proof. We prove (i). Let v be any column of U for which E has a zero column. Then, $LAv = 0 \implies L^{-1}LAv = L^{-1}0 \implies Av = 0$, hence $v \in \operatorname{Ker} A$, where we used that L is invertible. The invertibility of U guarantees that selecting s distinct columns from U spans an s-dimensional space. If we pick those s distinct columns such that the corresponding columns in E are zero, then this space must be $\operatorname{Ker} A$. The proof for (ii) follows similarly. $\square$

If $Ax = b$ is consistent, a solution is given by

$$x = UE^\intercal L\, b.$$

To check whether $Ax = b$ is consistent, we can use the inconsistency certificate from Giesbrecht *et al.* [GLS98]. This certificate requires a basis for the left nullspace of A. As mentioned before, the *LEU* decomposition computes such a basis anyway. In Section 4.3 we present a protocol that implements this inconsistency certificate.

4.1 Malaschonok's LEU Decomposition Algorithm

Protocol 2a and 2b show Malaschonok's algorithm in our notation.

Remark. Given a matrix $A \in \mathbb{K}^{m\times n}$ with arbitrary dimensions m and n, we can always pad this matrix with zeros (say, on the right and bottom side) to obtain a $2^d \times 2^d$ matrix, where $d := \lceil \log_2 \max(m, n)\rceil$.

Protocol 2 a $(\llbracket L\rrbracket, \llbracket E\rrbracket, U) \leftarrow \mathsf{LEUDecompose}(\llbracket A\rrbracket)$ (Base Case)

```
Input:  $A \in \mathbb{K}^{1\times 1}$
Output: $L, U \in \mathbb{K}, E \in \mathbb{B}$
1: $\llbracket b\rrbracket \leftarrow \llbracket A\rrbracket \overset{?}{=} 0$
2: $\llbracket x\rrbracket \leftarrow (\llbracket A\rrbracket + \llbracket b\rrbracket)^{-1}$
3: return $(\llbracket x\rrbracket, 1 - \llbracket b\rrbracket, 1)$
```

Protocol 2 b $(\llbracket L\rrbracket, \llbracket E\rrbracket, \llbracket U\rrbracket) \leftarrow \mathsf{LEUDecompose}(\llbracket A\rrbracket)$ (Recursive Step)

```
Input:  $A \in \mathbb{K}^{2^d\times 2^d}$ with $d \in \mathbb{N}, d > 0$
Output: $L \in \mathcal{L}_{2^d}, E \in \mathcal{P}_{2^d}, U \in \mathcal{U}_{2^d}$
1: $\begin{pmatrix}\llbracket A_{11}\rrbracket & \llbracket A_{12}\rrbracket\\ \llbracket A_{21}\rrbracket & \llbracket A_{22}\rrbracket\end{pmatrix} \leftarrow \llbracket A\rrbracket$    ▷ $A_{ij} \in \mathbb{K}^{2^{d-1}\times 2^{d-1}} \quad \forall i, j \in \{1, 2\}$
2: $(\llbracket L_{11}\rrbracket, \llbracket E_{11}\rrbracket, \llbracket U_{11}\rrbracket) \leftarrow \mathsf{LEUDecompose}(\llbracket A_{11}\rrbracket)$
3: $\llbracket Q\rrbracket \leftarrow \llbracket L_{11}\rrbracket\llbracket A_{12}\rrbracket, \qquad \llbracket B\rrbracket \leftarrow \llbracket A_{21}\rrbracket\llbracket U_{11}\rrbracket$
4: $(\llbracket L_{12}\rrbracket, \llbracket E_{12}\rrbracket, \llbracket U_{12}\rrbracket) \leftarrow \mathsf{LEUDecompose}((I - \llbracket E_{11}\rrbracket\llbracket E_{11}\rrbracket^{\mathsf{T}})\llbracket Q\rrbracket)$,
   $(\llbracket L_{21}\rrbracket, \llbracket E_{21}\rrbracket, \llbracket U_{21}\rrbracket) \leftarrow \mathsf{LEUDecompose}(\llbracket B\rrbracket(I - \llbracket E_{11}\rrbracket^{\mathsf{T}}\llbracket E_{11}\rrbracket))$
5: $\llbracket G\rrbracket \leftarrow \llbracket L_{21}\rrbracket(\llbracket A_{22}\rrbracket - \llbracket B\rrbracket\llbracket E_{11}\rrbracket^{\mathsf{T}}\llbracket Q\rrbracket)\llbracket U_{12}\rrbracket$
6: $\llbracket H\rrbracket \leftarrow (I - \llbracket E_{21}\rrbracket\llbracket E_{21}\rrbracket^{\mathsf{T}})\llbracket G\rrbracket(I - \llbracket E_{12}\rrbracket^{\mathsf{T}}\llbracket E_{12}\rrbracket)$
7: $(\llbracket L_{22}\rrbracket, \llbracket E_{22}\rrbracket, \llbracket U_{22}\rrbracket) \leftarrow \mathsf{LEUDecompose}(\llbracket H\rrbracket)$,
   $\llbracket V\rrbracket \leftarrow \llbracket U_{21}\rrbracket\llbracket E_{21}\rrbracket^{\mathsf{T}}\llbracket G\rrbracket(I - \llbracket E_{12}\rrbracket^{\mathsf{T}}\llbracket E_{12}\rrbracket) + \llbracket E_{11}\rrbracket^{\mathsf{T}}\llbracket Q\rrbracket\llbracket U_{12}\rrbracket$,
   $\llbracket W\rrbracket \leftarrow \llbracket G\rrbracket\llbracket E_{12}\rrbracket^{\mathsf{T}}\llbracket L_{12}\rrbracket + \llbracket L_{21}\rrbracket\llbracket B\rrbracket\llbracket E_{11}\rrbracket^{\mathsf{T}}$
8: $\llbracket L\rrbracket \leftarrow \begin{pmatrix}\llbracket L_{12}\rrbracket\llbracket L_{11}\rrbracket & 0\\ -\llbracket L_{22}\rrbracket\llbracket W\rrbracket\llbracket L_{11}\rrbracket & \llbracket L_{22}\rrbracket\llbracket L_{21}\rrbracket\end{pmatrix}, \qquad \llbracket E\rrbracket \leftarrow \begin{pmatrix}\llbracket E_{11}\rrbracket & \llbracket E_{12}\rrbracket\\ \llbracket E_{21}\rrbracket & \llbracket E_{22}\rrbracket\end{pmatrix}$,
   $\llbracket U\rrbracket \leftarrow \begin{pmatrix}\llbracket U_{11}\rrbracket\llbracket U_{21}\rrbracket & -\llbracket U_{11}\rrbracket\llbracket V\rrbracket\llbracket U_{22}\rrbracket\\ 0 & \llbracket U_{12}\rrbracket\llbracket U_{22}\rrbracket\end{pmatrix}$
```

4.2 Complexity Analysis

Computational Complexity. We derive an expression for the computational complexity under the assumption that secure matrix multiplication costs $O(n^2)$ secure inner products. For example, this is the case when we use Shamir's linear secret sharing scheme as the underlying MPC scheme.

Theorem 4. *Let $n = 2^d$ for $d \in \mathbb{N}$. For any $n \times n$ matrix over $\mathbb{K}$, Protocol 2 requires*

$$C(n) = 25 \left(\frac{n}{2}\right)^2 \log_2 n + 5\binom{n}{2} + n^2\alpha$$

secure inner-products, where $\alpha \in \mathbb{N}$ represents the number of secure inner-products required for executing the base case of Protocol 2 (i.e., the number of secure inner-product invocations required to perform a secure equality test and inverting a secret-shared field element).

Proof. Let $m := n/2$. We start by counting the number of secure inner products per line in Protocol 2b:

line no	# secure inner-products
3	$2m^2$
4	$2(m + m^2)$
5	$4m^2$
6	$2(m + m^2)$
7	$m + 9m^2$
8	$2m^2 + n^2 = 6m^2$

In total, the algorithm uses $25m^2 + 5m$ secure inner products per recursion step, and at each recursion level the algorithm invokes itself four times. This gives rise to the following recurrence equation

$$\tilde{C}(d) = 25 \cdot 2^{2(d-1)} + 5 \cdot 2^{d-1} + 4\,\tilde{C}(d-1)$$

with initial condition $\tilde{C}(0) = \alpha$. We now claim that the bound in the theorem statement, $C(n)$, is the solution to this equation with $\tilde{C}(d) = C(2^d)$. First, we check the initial condition, and indeed $\tilde{C}(0) = C(1) = \alpha$. Assume the statement holds for d. Then, by induction, we have

$$\begin{aligned}
\tilde{C}(d+1) = C(2n) &= 25 \left(\frac{2n}{2}\right)^2 \log_2 2n + 5\binom{2n}{2} + (2n)^2\alpha \\
&= 4 \cdot 25 \left(\frac{n}{2}\right)^2 (1 + \log_2 n) + 5\left(\frac{(2n)^2}{2} - \frac{2n}{2}\right) + 4n^2\alpha \\
&= 25n^2 + 4 \cdot 25 \left(\frac{n}{2}\right)^2 \log_2 n + 5 \cdot 4\left(\frac{n^2}{2} - \frac{n}{2} + \frac{n}{4}\right) + 4n^2\alpha \\
&= 25n^2 + 5n + 4 \cdot \left(25\left(\frac{n}{2}\right)^2 \log_2 n + 5\binom{n}{2} + n^2\alpha\right) \\
&= 25n^2 + 5n + 4\,C(n),
\end{aligned}$$

which proves the claim. □

Round Complexity. As for the round complexity, we merely aim for an asymptotic expression, because it is rather cumbersome to find the optimal schedule of operations that would lead to the exact minimum number of rounds.

Theorem 5. *Let $m = 2^b$ for $b \in \mathbb{N}$, and suppose that Protocol 2 requires $B \in \mathbb{N}$ rounds of communication when decomposing an arbitrary $m \times m$ matrix over $\mathbb{K}$. Then, for any $n \times n$ matrix over $\mathbb{K}$, $n = 2^d$ with $d \in \mathbb{N}$ such that $d > b$, Protocol 2 requires*

$$R(n) = n^{\log_2(3)} \frac{1}{3^b} \left(B + \frac{\gamma}{2} \right) - \frac{\gamma}{2} = O(n^{1.585}), \qquad n > 2^b$$

rounds of communication, where $\gamma \in \mathbb{N}$ represents the number of rounds of communication required to execute one call to Protocol 2b when excluding the recursive calls.

Proof. Protocol 2b makes four recursive calls to itself, where we note that the second and third call can be executed in parallel. This gives rise to the following recurrence equation:

$$R(d) = 3R(d-1) + \gamma,$$

with initial condition $R(b) = B$. Solving this equation yields the claim. $\square$

4.3 Inconsistency Certificate

Protocol 3 shows a simple protocol for computing the inconsistency certificate of Giesbrecht *et al.* [GLS98]. The basic idea behind this certificate is as follows: if $Ax = b$ is inconsistent, then $\dim \mathrm{Ker}([A|b]^\mathsf{T}) = \dim \mathrm{Ker}(A^\mathsf{T}) - 1$. Hence, if we sample a random vector v from the left null space of A, with high probability $(1 - \varepsilon \geq 1 - |\mathbb{K}|^{-1})$ this vector does not lie in $\mathrm{Ker}([A|b]^\mathsf{T})$ and in this case $v \cdot b$ will be nonzero. In the consistent case, b lies in the column space of A, hence $v \cdot b$ will always vanish. The error probability ε follows from the Schwartz–Zippel Lemma; for details we refer to Theorem 2.1 and 2.2 in [GLS98].

Complexity. InconsistencyCertificate samples n public random field elements, and performs n secure multiplications, 1 secure inner-product and 1 secure zero test, where n is the size of A.

Bibliography

[Bar68] Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Mathematics of Computation*, 22(103):565–578, 1968.

Protocol 3 $[\![c]\!] \leftarrow \mathsf{InconsistencyCertificate}([\![A]\!], [\![b]\!])$

Input: $A \in \mathbb{K}^{2^d \times 2^d}, b \in \mathbb{K}^{2^d}$

Output: $c \in \{0,1\} \subset \mathbb{K}$, with the following interpretation: if $c = 1$, then the system $Ax = b$ is inconsistent (with certainty). If $c = 0$, the system is consistent except with probability $\varepsilon \leq |\mathbb{K}|^{-1}$.

```
1: $(L, E, U) \leftarrow \mathsf{LEUDecompose}(A)$
2: Sample $(\alpha_1, \ldots, \alpha_n) \leftarrow \mathbb{K}^n$ uniformly and independently at random
3: for $i \leftarrow 1$ to $n$ do
4:     $[\![\mu_i]\!] \leftarrow 1 - \sum_{j=1}^{n} [\![E_{i,j}]\!]$
5:     for $j \leftarrow 1$ to $n$ do
6:         $[\![v_j]\!] \leftarrow [\![v_j]\!] + \alpha_i [\![\mu_i]\!] [\![L_{i,j}]\!]$
7: $[\![s]\!] \leftarrow [\![v]\!] \cdot [\![b]\!]$                    ▷ $v = (v_1, \ldots, v_n)$
8: return $1 - ([\![s]\!] \overset{?}{=} 0)$
```

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[BIB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proc. 8th Symp. on Princip. of Distr. Comp.*, pages 201–209, NY, 1989. ACM.

[CD01] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In *Proc. CRYPTO 2001, Santa Barbara, USA*, pages 119–136. Springer, 2001.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *Eurocrypt 2001, Innsbruck, Austria*, volume 2045 of *LNCS*, pages 280–299. Springer, 2001.

[CDN15] Ronald Cramer, Ivan Damgard, and Jesper Buus Nielsen. *Secure multiparty computation and secret sharing-an information theoretic approach.* Cambridge, 2015.

[CKP07] Ronald Cramer, Eike Kiltz, and Carles Padró. A note on secure computation of the Moore–Penrose pseudoinverse and its application to secure linear algebra. In *Proc. CRYPTO 2007, Santa Barbara, USA*, pages 613–630. Springer, 2007.

[DN03] Ivan Damgård and Jesper Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. *Advances in Cryptology-CRYPTO 2003*, pages 247–264, 2003.

[DPS15] Jean-Guillaume Dumas, Clément Pernet, and Ziad Sultan. Computing the rank profile matrix. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 149–156. ACM, 2015.

[GLS98] Mark Giesbrecht, Austin Lobo, and B David Saunders. Certifying inconsistency of sparse linear systems. In *Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pages 113–119. ACM, 1998.

[JPS13] Claude-Pierre Jeannerod, Clément Pernet, and Arne Storjohann. Rank-profile revealing Gaussian elimination and the CUP matrix decomposition. *Journal of Symbolic Computation*, 56:46–68, 2013.

[KL96] E. Kaltofen and A. Lobo. On rank properties of toeplitz matrices over finite fields. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, ISSAC '96, pages 241–249, New York, NY, USA, 1996. ACM.

[KS91] Erich Kaltofen and B David Saunders. On Wiedemann's method of solving sparse linear systems. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 29–38. Springer, 1991.

[Mal10] Gennadi Malaschonok. Fast generalized Bruhat decomposition. In *International Workshop on Computer Algebra in Scientific Computing*, pages 194–202. Springer, 2010.

[NO07] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007, Beijing, China*, volume 4450 of *LNCS*, pages 343–360. Springer, 2007.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.