D2.3 Use-Case Specific Algorithms

Mark Abspoel (PHI), Frank Blom (TUE), Niek J. Bouman (TUE),
Tore Frederiksen (ALX), Paul Koster (PHI), Claudio Orlandi (AU),
Mykola Pechenizkiy (TUE), Berry Schoenmakers (TUE),
Meilof Veeningen (PHI), Nikolaj Volgushev (ALX), Niels de Vreede (TUE)

The project SODA has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731583.

Project Information

Scalable Oblivious Data Analytics

Project number:	731583
Strategic objective:	H2020-ICT-2016-1
Starting date:	2017-01-01
Ending date:	2019-12-31
Website:	https://www.soda-project.eu/

Document Information

Title: D2.3 Use-Case Specific Algorithms

ID:	D2.3	Type:	R	Dissemination level:	PU
		Month:	M33	Release date:	2019-09-30

Contributors, Editor & Reviewer Information

Contributors (person(partner): sections)	Mark Abspoel (PHI): Chapter 6 Frank Blom (TUE): Chapter 2 Niek J. Bouman (TUE): Chapter 1 Tore Frederiksen (ALX): Chapter 5 Paul Koster (PHI): Chapter 3 Claudio Orlandi (AU): Chapter 4 Mykola Pechenizkiy (TUE): Chapter 1 Berry Schoenmakers (TUE): Chapter 2 Meilof Veeningen (PHI): Chapter 3 Nikolaj Volgushev (ALX): Chapters 5,6 Niels de Vreede (TUE): Chapter 1
Editor (person/partner)	Niek J. Bouman (PHI)
Reviewers (person/partner)	Paul Koster (PHI), Peter Scholl (AU)

Release number	Date issued	SVN version	Release description / changes made
1	2019-09-30	852	Initial release

SODA Consortium

Full Name	Abbreviated Name	Country
Philips Electronics Nederland B.V.	PHI	Netherlands
Alexandra Institute	ALX	Denmark
Aarhus University	AU	Denmark
Göttingen University	GU	Germany
Eindhoven University of Technology	TUE	Netherlands

Table 1: Consortium Members

Executive summary

This WP2 deliverable contains research results on the topic of secure multiparty computation that have been obtained during the second half of the SODA project. The overall goal of SODA is to devise optimized multiparty computation protocols that enable or make a step forward towards the application of MPC to large volumes of data. The deliverable contributes to this goal in various directions; it contains new results on secure linear algebra, protocols in the streaming setting, new MPC protocols for (obliviously) applying a function to certain elements of a sparse data set, with applications to Kaplan–Meier survival analysis, on combining two-party computation with private set intersection, and obliviously training a decision tree on a secret-shared database with continuous attributes. Below, we give a brief summary per chapter.

A Practical Approach to the Secure Computation of the Moore–Penrose Pseudoinverse over the Rationals. This chapter presents a practical protocol for computing the Moore–Penrose pseudoinverse in a linear number of rounds (linear in the dimensions of the input matrix) over the rational numbers, by means of solving the corresponding problem over a finite field. The main practical application of this work is to securely solve systems of linear equations of which the rank is unknown.

Secure Streaming Algorithms for Sums of Frequency Moments. A *data stream* is an ordered sequence of elements, possibly of infinite length, that can only be accessed sequentially. Data streams are a powerful abstraction when processing very large amounts of data.

In this chapter we devise MPC protocols in the streaming setting for computing sums of frequency moments, inspired by the well-known streaming algorithms due to Alon, Matias and Szegedy [2].

Efficient Private Map on Sparse Datasets & Kaplan-Meier Survival Analysis. This chapter presents efficient and private "filtered map" and "filtered map–reduce" operations, where the latter is applied to Kaplan-Meier survival analysis. Specifically, the method optimizes the speed of the logrank test under MPC to decide if two Kaplan-Meier estimates are statistically different.

Combining Private Set-Intersection with Secure Two-Party Computation. The problem of Private Set-Intersection (PSI) is a popular topic in secure two-party computation (2PC). Unfortunately, PSI protocols can typically not be plugged into larger 2PC applications since in these protocols one party (by design) learns the output of the intersection. This chapter describes a novel and efficient OT-based PSI protocol that produces an "encrypted" output that can be securely post-processed with other 2PC protocols.

Oblivious Querying of a Distributed Database In this chapter we consider how to allow parties to do outsourced MPC on a few elements from a secret-shared database. Our specific focus in this chapter is on how to achieve this with minimal communication and computation of the parties accessing the secret-shared database. We show how to achieve this only relying on simple, symmetric cryptographic primitives, thus allowing for an implementation that could be deployed in an app or as a web-service.

Private Training of Decision Trees with Continuous Attributes This chapter is an abstract on work where we show how to train a decision tree securely from a database that is secret-shared among multiple mutually distrustful parties. We consider data with continuous (i.e. real-valued) attributes, and

develop a secure version of a learning algorithm similar to the C4.5 or CART algorithms. We obtain a protocol, which we also implement, that obtains practical efficiency for large datasets.

About this Document

Role of the deliverable

This deliverable contains several research results on the topic of MPC tailored to the problems in data mining, and have been obtained during the second half of the SODA project. This deliverable is part of Work Package 2. The scope of this deliverable is application-oriented, with a strong focus on data mining applications. More precisely, our scope includes general building blocks relevant for data mining (such as linear algebra), as well as application-specific research work (e.g., privacy-preserving medical survival analysis).

Within this scope, our objective is to devise optimized multi-party computation protocols that enable or make a step forward towards the application of MPC to large volumes of data. In the light of the above objective, this deliverable presents several contributions, i.e., protocols for securely solving linear systems with unknown rank via the Moore–Penrose pseudoinverse, protocols for securely estimating the number of distinct secret-shared elements in a stream, realizations of the "map" and "map–reduce" functional-programming paradigms in MPC, with applications to privacy-preserving Kaplan–Meier survival analysis, combining two-party private set intersection with secure post-processing, obliviously querying a distributed secret-shared database, and obliviously training a decision tree on a secret-shared database with continuous attributes.

Relationship to other SODA deliverables

This deliverable builds upon the deliverables D1.1, D2.1, which survey the state of the art in privacy-preserving data mining, as well as upon deliverables D1.2 and D2.2, which contain the research output obtained during the first half of the SODA project.

In particular, Chapter 1 from this deliverable generalizes the linear-algebra results from the first chapter of D2.2 to a setting in which the rank of the secret-shared input matrix is unknown. In Chapter 3, the efficient private map method is applied to the Kaplan–Meier survival rate analysis, which was introduced in SODA deliverable D4.1 and is to be implemented in D4.5. Similarly, the work on decision trees from Chapter 6 serves as a basis for one of the medical predictive modelling demonstrators in D4.5.

Relationship to other versions of this deliverable

N/A

Structure of this document

The document contains six chapters covering recent research. Chapter 1 presents a new protocol for securely solving unknown-rank linear systems over the rationals via the Moore–Penrose pseudoinverse. Chapter 2 proposes protocols for securely computing sums of frequency moments, a fundamental computational task in the streaming-model-of-computation literature. Chapter 3 introduces protocols for securely performing "map" and "map–reduce" operations, which are well-known computation paradigms from functional programming. The map procotol is applied to Kaplan–Meier survival rate analysis. Chapter 4 focuses on the problem of two-party private-set-intersection with secure post-processing, and proposes a private-set-intersection protocol that produces an "encrypted" result, which can be further post-processed as a two-party secure computation. Chapter 5 proposes a new protocol for performing outsourced MPC on a few elements that are retrieved obliviously from a

secret-shared database. Chapter 6 gives a protocol and implementation for securely training a decision tree on a secret-shared dataset with continuous attributes.

Table of Contents

Chapter 1

A Practical Approach to the Secure Computation of the Moore–Penrose Pseudoinverse over the Rationals

1.1 Introduction

Motivated by the goal of performing elementary statistical tasks such as linear regression *securely*, we revisit the topic of secure linear algebra. In this paper, "securely" refers to *secure multiparty computation* (MPC) [25], however, our results might be of use in other settings as well, for example, for mitigating certain side-channel attacks in trusted execution environments in CPUs.

Secure linear algebra goes back to the work of Cramer and Damgård [23], who proposed constant-rounds MPC protocols for various basic tasks in linear algebra. In that paper, as well as in later papers in the same line of work, like [84, 66, 26, 75], the focus is on linear algebra *over a finite field*.

Our goal is to obtain, in an "MPC-friendly" way, an (approximate) solution to a linear system *over the real numbers*. In this paper we choose to approximate real arithmetic by (exact) rational arithmetic, or, in fact, integer arithmetic, using appropriate scaling. Our main reason behind this choice is the close connection between the finite field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ (where p is prime) and integer arithmetic, since we target MPC schemes that offer finite-field arithmetic. Hence, the protocols that we propose in this paper will employ finite-field arithmetic *as a tool, rather than as a goal*. We note that there are various papers targeting the same problem that explore other choices, such as secure fixed-point arithmetic (see, e.g., [82, 46]) or secure floating-point arithmetic (e.g., [14]).

In an earlier joint work with Blom and Schoenmakers [13], we focused on the case of solving full-rank systems; in this paper we deal with the unknown-rank case. Also, we would like to obtain meaningful solutions in case the system is over- or underdetermined. The *Moore–Penrose pseudoinverse* gives natural solutions in both cases: in the overdetermined case, which is the relevant case for linear regression, it yields the least-squares solution; in the underdetermined case it gives the minimum-norm solution.

Concretely, given a matrix A with integral elements of unknown rank, we propose a protocol for computing the Moore–Penrose pseudoinverse over the rational numbers in a linear number of rounds. The computational complexity, counted as the number of secure multiplications, is $O(m^2n)$, where m and n, $m \leq n$, are the dimensions of the system. In multiplicative-linear-secret-sharing-based MPC schemes, such as Shamir's scheme, we may count a secure inner product as a single secure multiplication; in that case the complexity reduces to $O(mn)$.

It should be rather easy to implement our protocol in any finite-field-based arithmetic secret-sharing MPC framework; beyond elementary finite-field arithmetic our protocol merely requires secure subprotocols for sampling (public) random elements, performing a zero test on a secret-shared field element, computing the reciprocal of a secret-shared field element, and computing the determinant of an invertible secret-shared matrix.

Circumventing Rational Reconstruction. It is well known that one can perform (bounded) rational arithmetic via arithmetic in $\mathbb{F}_p$, essentially as follows: (i) represent the rational inputs as finite-field elements, i.e., an input of the form a/b, for integers a and b and such that $|a|,|b| \leq \sqrt{p/2}$, is encoded as the element $x = a \cdot b^{-1} \in \mathbb{F}_p$, (ii) perform the computation in integer arithmetic modulo p, (iii) reconstruct the numerators and denominators of the results of the computation, elementwise, in the following manner. Let $y \in \mathbb{F}_p$ be an output of the computation, that corresponds to the fraction c/d for integers c and d. Then, if $|c|,|d| \leq \sqrt{p/2}$, we can uniquely reconstruct c and d from y by reducing the two-dimensional lattice basis $\{(p,0),(y,1)\}$ using the Lagrange–Gauss algorithm, in the sense that the reduced basis will contain the vector (c,d). This reconstruction procedure is known as *rational reconstruction* (see, e.g., [102])

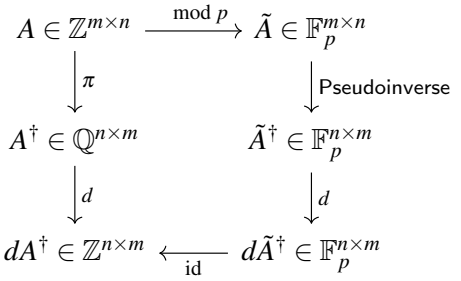
An important drawback of the use of rational reconstruction in our scenario is that we essentially would need to *double* the bit-length of the finite field modulus p to guarantee unique reconstruction, compared to a route without rational reconstruction (for more details, see Figure 1.1). Because arithmetic in a larger finite field is computationally more expensive, we would like to avoid the use of rational reconstruction.

In [13], a key trick for obtaining the inverse of an invertible integer matrix B over the rational numbers from the corresponding inverse over the finite field $\mathbb{F}_p$ *without requiring rational reconstruction*, was to form the integer-valued *adjugate matrix* by multiplying B^{-1} by $\det B$. In a similar spirit, we compute the pseudoinverse $A^\dagger$ over the finite field $\mathbb{F}_p$ and identify the conditions under which it corresponds to the pseudoinverse over the rational numbers. Essentially, this comes down to choosing p sufficiently large; see Section 1.4.2. We can then obtain an integer representation of the pseudoinverse by forming the pair $(dA^\dagger, d)$, where $dA^\dagger$ is an integer matrix containing the numerators of the pseudoinverse and d is the common denominator of the pseudoinverse, which coincides with the squared *volume* of A [10], which we write as $(\mathrm{vol}A)^2$. Figure 1.1 illustrates our approach and compares it to the alternative route of rational reconstruction.

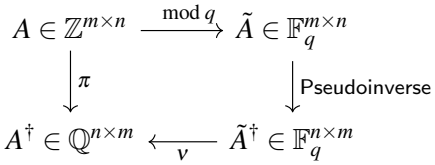
Although taking the square of the volume is rather excessive in certain cases (for example, the magnitude of the common denominator of B^{-1}, for any *invertible* matrix B, equals $|\det B| = \mathrm{vol}B$), it is essentially the price we have to pay for not knowing whether we are dealing with such a special case.

Computing the Pseudoinverse and Its Common Denominator. To compute the Moore–Penrose pseudoinverse of A obliviously, we first compute a *reflexive generalized inverse* of the symmetric product $AA^\mathsf{T}AA^\mathsf{T}$ by means of block-recursive elimination. We then compute the Moore–Penrose pseudoinverse from this generalized inverse.

Regarding the common denominator, Springer computes $(\mathrm{vol}A)^2$ via an integer-preserving rank decomposition [99]. To circumvent the need for constructing such a rank decomposition, we seek a simpler alternative. Ben-Israel gives a method for computing $(\mathrm{vol}A)^2$ that requires an orthonormal basis for the left nullspace of A [10]. Although an *orthonormal* basis might not even exist over a finite field, we can easily construct a matrix K whose columns span the left nullspace of A. We generalize Ben-Israel's result so that we can compute $(\mathrm{vol}A)^2$ from A and K.



(a) Our approach. The map d represents scalar multiplication by $d = (\mathrm{vol}A)^2$ and id represents the identity map. The solutions $dA^\dagger$ and $d\tilde{A}^\dagger$ coincide, provided that p is chosen large enough, i.e., according to the theorem in Section 1.4.2.



(b) Approach using rational reconstruction. The map ν represents the elementwise rational reconstruction procedure. All reconstructed fractions will be in lowest terms (numerator and denominator have no common nontrivial factors). There is, however, a price to be paid, in that $q \geq 2p^2$. Also, the map ν (the Lagrange–Gauss algorithm) is not "MPC-friendly".

Figure 1.1: Comparison between our approach and the approach via rational reconstruction. In the diagrams, the map $\pi : \mathbb{Q}^{m\times n} \to \mathbb{Q}^{n\times m}, A \mapsto A^\dagger$ applies the Moore–Penrose inverse over the rationals.

Preconditioning for Computing Pseudoinverses. As noted above, we will compute the Moore–Penrose inverse via a generalized inverse that is obtained using block-recursive elimination.

Deterministic elimination algorithms typically employ pivoting to avoid problems like division by zero. Pivoting involves searching for and applying suitable row and/or column swaps prior to each elimination step. In secure computation, however, we aim to avoid pivoting because searching for particular elements and applying data-dependent row and column swaps, *obliviously*, is expensive (in a computational- and round-complexity sense).

An MPC-friendly alternative is to transform the matrix to be eliminated into an equivalent matrix for which the elimination procedure will succeed *without any pivoting*; this approach is called *preconditioning*. In case of Gaussian elimination, for example, the condition of *generic rank profile*[1] guarantees that pivoting can be omitted. As we prove in this paper, generic rank profile is also a sufficient condition for correctness of the particular block-recursive elimination algorithm that we use.

When dealing with a square, full rank matrix B over a finite field $\mathbb{F}$ with large order, one way to achieve generic rank profile with high probability is by pre-multiplying B by a preconditioner matrix R that is chosen uniformly at random from the set of all invertible matrices having the same size as B. When computing the inverse of RB, we can apply the rule $(RB)^{-1} = B^{-1}R^{-1}$, which we will refer to as the *reverse order law* for matrix inversion, to show that the inverse of the preconditioner can easily be removed by post-multiplying by R. For a matrix A with arbitrary rank r, pre-multiplying by a randomly chosen invertible matrix R (of appropriate size) is not sufficient for achieving generic rank profile; we additionally need to mix A's columns by multiplying A by a preconditioner matrix from the right.

A major problem that arises when trying to remove a preconditioner when computing the pseudoinverse, is that the reverse order law for pseudoinverses does not hold in general [48, 50]. In particular, unfortunately, we have that $(LAR)^\dagger$ does not necessarily equal $R^\dagger A^\dagger L^\dagger$ for invertible preconditioner matrices L and R. Hence, we cannot simply extract $A^\dagger$ from $(LAR)^\dagger$ like we could do above for B^{-1}. We circumvent this problem by removing the preconditioner immediately after computing the reflexive generalized inverse, for which the reverse-order law does hold.

[1] A matrix A of rank r has *generic rank profile* if and only if all upper-left square submatrices of A up to dimension $r \times r$ are invertible.

An additional constraint in our setting is that the preconditioner should preserve symmetry, since the symmetry property enables significant computational savings during elimination. A preconditioner for this particular scenario seems to be lacking in the literature. We resolve this by proving that the preconditioner $A \mapsto UAU^{\mathsf{T}}$ for a uniformly random matrix U fulfills all our constraints.

Interestingly, and unlike Gaussian elimination, when working over the real or complex numbers, the particular block-recursive algorithm that we use for computing the reflexive generalized inverse does not even require its input to have generic rank profile, hence no preconditioning is needed in this case. Nonetheless, in fields with positive characteristic, the condition emerges from the phenomenon of self-orthogonality.

1.1.1 Related Work

Cramer, Kiltz and Padró [26] propose a constant-rounds protocol for securely computing the Moore–Penrose pseudoinverse over a finite field. Their approach is to first compute the characteristic polynomial of the Gram matrix $A^{\mathsf{T}}A$, from which they then compute the rank of A (via a technique by Mulmuley [79]) as well as the pseudoinverse of A (via the Cayley–Hamilton theorem).

An important theme in [26] is to ensure that A (and A^{T}) are *suitable*, which guarantees, informally speaking, that certain subspaces that are orthogonal over a field with characteristic zero, remain orthogonal over fields with positive characteristic. In our work, where we focus on the setting where the modulus (hence the field's characteristic) is chosen sufficiently large, existence of the pseudoinverse is guaranteed by a result in [6]. (We state this result in the next section.) Nonetheless, as described in the previous section, we do take special precautions, namely, applying preconditioning, to avoid problems related to working over a field with positive characteristic when computing a reflexive generalized inverse.

For an $m \times n$ matrix where $m \leq n$, the complexity (number of secure multiplications) of Cramer et al.'s solution is $O(m^4 + n^2 m)$. Our solution, albeit not constant-rounds, has complexity $O(m^2 n)$, and even $O(mn)$ when assuming availability of a "cheap inner product", where the hidden constants in the Big-Oh of our solution are single-digit integers. By "cheap inner product", we mean that an inner product between two vectors of the same but arbitrary length has the same communication and round complexity as a single secure multiplication. It is possible to perform multiplication of an $m \times \ell$ matrix by an $\ell \times n$ matrix using no more than mn "cheap inner products". Because the coefficients of the result matrix may all be mutually independent, it is reasonable to take the complexity of such a matrix product to be equal to mn.

We leave it to further research to compare the practical performance of our method to that of [26] in various application scenarios (i.e., various matrix-dimension regimes, network latency, bounded computational resources and storage space, etc.).

Relation to the LEU Decomposition. An earlier work by the authors [18] proposes to use Malaschonok's *LEU* decomposition [71] for solving linear systems of arbitrary rank in the context of secure computation. (Note that [18] does not deal with the problem of computing the Moore–Penrose pseudoinverse.) Our new protocol Pseudoinverse is superior to the *LEU*-decomposition-based protocol; in terms of round complexity, $O(m)$ versus $O(m^{1.59})$, as well as in terms of the asymptotic computational complexity, $O(m^2)$ versus $O(m^2 \log m)$ secure inner products for a square $m \times m$ matrix.

1.2 Preliminaries

Secret Sharing and Secure Computation. Let $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$. We use $\mathbb{F}$ to denote an arbitrary field. We assume the use of an MPC protocol based on arithmetic secret-sharing over $\mathbb{F}_p$. Our protocols will inherit the security properties (passive vs. active) from the underlying MPC protocol and of the subprotocols invoked by our protocol. The notation $[\![x]\!]$ represents an element $x \in \mathbb{F}_p$ that is secret-shared among the parties in the MPC protocol. Notation for secure arithmetic then follows naturally, for example, $[\![c]\!] \leftarrow [\![a]\!] + [\![b]\!]$ describes the addition of a and b where the result is stored in a new secret-shared element c, and $[\![d]\!] \leftarrow [\![a]\!][\![b]\!]$ describes an invocation of the multiplication protocol to securely compute the product of a and b and store the result in d. For arbitrary integer matrices A and B, the notation $[\![A]\!]$ expresses that all elements of A are secret-shared over $\mathbb{F}_p$, and $[\![A]\!] + [\![B]\!]$ and $[\![A]\!][\![B]\!]$ represent secure matrix addition (which coincides with elementwise addition) and secure matrix multiplication, respectively. Our protocols assume the availability of subprotocols for securely sampling private as well as public random field elements (e.g., [24]), denoted as $[\![a]\!] \xleftarrow{\$} \mathbb{F}_p$ and $a \xleftarrow{\$} \mathbb{F}_p$ respectively, for securely inverting a field element (see [7]), and for performing a secure zero test [30, 83]. The latter two are denoted as protocols Reciprocal and IsZero, respectively. We require protocol Reciprocal to be secure for all nonzero inputs (i.e., the protocol is allowed to leak information when run on a secret share of zero). Protocol IsZero returns $[\![1]\!]$ if its argument equals zero and returns $[\![0]\!]$ otherwise.

Generalized Inverses. A *generalized inverse* of a matrix A is a matrix X associated to A that exists for a class of matrices larger than the class of invertible matrices, shares some properties with the ordinary inverse, and reduces to the ordinary inverse when A is non-singular. In this paper, we classify generalized inverses using the following four properties, also known as the *Penrose equations*:

$$(1)\, AXA = A, \quad (2)\, XAX = X, \quad (3)\, (AX)^{\mathsf{T}} = AX, \quad (4)\, (XA)^{\mathsf{T}} = XA.$$

The matrix X that satisfies all four Penrose equations for a given matrix A is called the *Moore–Penrose pseudoinverse*, or simply *pseudoinverse* of A, which we denote as $A^{\dagger}$. The Moore–Penrose inverse of A over $\mathbb{F}$ exists if and only if $\operatorname{rank}(AA^{\mathsf{T}}) = \operatorname{rank}(A^{\mathsf{T}}A) = \operatorname{rank} A$ [87, Thm 1], and if it exists it is unique. We will also focus on generalized inverses of A which only satisfy equations (1) and (2); such generalized inverses are called *reflexive generalized inverses* and we denote any reflexive generalized inverse of A by A^{-}. Note that reflexive generalized inverses are not necessarily unique. For an extensive treatment of generalized inverses, the reader is referred to [11].

For a square matrix A partitioned as

$$A = \begin{pmatrix} E & F \\ G & H \end{pmatrix} \tag{1.5}$$

such that E is square, A/E denotes the *generalized Schur complement*

$$A/E = H - GE^{-}F.$$

Submatrices, Their Determinants and Rank Properties. For any $n \in \mathbb{N}$, we write $[n]$ for the set $\{1, \ldots, n\}$. For any $m \times n$ matrix A and index sets $\mathscr{I} \subset [m]$ and $\mathscr{J} \subset [n]$, $[A]_{\mathscr{I},\mathscr{J}}$ denotes the determinant of the submatrix of A obtained by selecting all rows in $\mathscr{I}$ and all columns in $\mathscr{J}$. Furthermore, $A_{[k]}$ denotes the leading principal submatrix of order k, i.e., the matrix obtained by taking the first k

rows and first k columns of A, and we use $[A]_k$ as shorthand for $[A]_{[k],[k]}$, i.e., the leading principal minor of order k. Thus, it holds that $\det A_{[k]} = [A]_k$.

Let A be a matrix of rank r. We say that a matrix A has *generic rank profile* [59] if for all $k \in [r]$, it holds that A's leading principal minor of order k is nonzero.

Let A be partitioned as in (1.5). If $\det E \neq 0$, then *Schur's determinant formula* asserts that

$$\det A = \det(E)\det(A/E) = \det(E)\det(H - GE^{-1}F).$$

A direct consequence of [72, Thm 19] is that

$$\operatorname{rank} A \geq \operatorname{rank} E + \operatorname{rank}(A/E).$$

Hence, if A has generic rank profile and E has at least dimension $r \times r$ where $r = \operatorname{rank} A$, then A/E is the null matrix.

The Volume of a Matrix. For any matrix A with rank r and nonzero singular values $\sigma_1, \ldots, \sigma_r$, its *volume* is defined as $\operatorname{vol} A = \prod_{i=1}^{r} \sigma_i$. Note that this definition implies that we define the volume of the zero matrix to be one, which will be convenient for our purpose but deviates from Ben-Israel's definition of matrix volume for this special case [10]. A matrix over an integral domain has a pseudoinverse if and only if its squared volume is a unit (i.e., an invertible element) of the integral domain [6]. The fact that, for any matrix $A \in \mathbb{R}^{m \times n}$, the singular values of AA^{T} are the squares of the singular values of A leads to the following equation:

$$\operatorname{vol}(AA^{\mathsf{T}}) = (\operatorname{vol} A)^2, \tag{1.6}$$

which holds over an arbitrary field. In case A is a square nonsingular matrix, i.e., $m = n$ and $\det A \neq 0$, its volume coincides with the absolute value of its determinant:

$$\operatorname{vol} A = |\det A|. \tag{1.7}$$

Combining the two preceding equations gives

$$(\operatorname{vol} A)^2 = \det(AA^{\mathsf{T}}), \tag{1.8}$$

in the case that $\operatorname{rank} A = m$.

1.3 Block-Recursive Elimination

In this section we present ObliviousRGInverse, our oblivious protocol for computing a reflexive generalized inverse of any symmetric matrix over $\mathbb{F}_p$ that has generic rank profile. Although we could easily devise a protocol that also works for non-symmetric matrices, we deliberately restrict to symmetric matrices, for the following two reasons: (i) by doing so, we achieve a significant computational saving (essentially a factor of two); and (ii) for our application we anyway only need to compute a reflexive generalized inverse of a symmetric matrix.

First, we define the *extended reciprocal* of an element $c \in \mathbb{F}$ as zero if $c = 0$ and c^{-1}, i.e., the (ordinary) reciprocal, otherwise. Note that the reflexive generalized inverse of a 1×1 matrix is equal to the 1×1 matrix containing the extended reciprocal of its only coefficient. ScalarRGInverse is a secure protocol for computing the extended reciprocal. ObliviousRGInverse is given as Protocol 2. On line 4, the partitioning is done such that E and G are square and their dimensions differ by at most one. We remark that the side notes with label "symmetric" in ObliviousRGInverse indicate that the

Protocol 1 ScalarRGInverse($[\![a]\!]$)

1: $[\![z]\!] \leftarrow \mathsf{IsZero}([\![a]\!])$
2: **return** $\mathsf{Reciprocal}([\![a+z]\!]) - [\![z]\!]$

Protocol 2 ObliviousRGInverse($[\![A]\!]$)

1: **if** $n = 1$ **then**
2: **return** $\mathsf{ScalarRGInverse}([\![a_{1,1}]\!])$
3: **else**
4: $\begin{pmatrix} [\![E]\!] & [\![F]\!] \\ [\![F^\mathsf{T}]\!] & [\![G]\!] \end{pmatrix} \leftarrow [\![A]\!]$ ▷ split as evenly as possible
5: $[\![X]\!] \leftarrow \mathsf{ObliviousRGInverse}([\![E]\!])$
6: $[\![XF]\!] \leftarrow [\![X]\!][\![F]\!]$
7: $[\![G - F^\mathsf{T}XF]\!] \leftarrow [\![G]\!] - [\![F^\mathsf{T}]\!][\![XF]\!]$ ▷ symmetric
8: $[\![Y]\!] \leftarrow \mathsf{ObliviousRGInverse}([\![G - F^\mathsf{T}XF]\!])$
9: $[\![XFY]\!] \leftarrow [\![XF]\!][\![Y]\!]$
10: $[\![X + XFYF^\mathsf{T}X]\!] \leftarrow [\![X]\!] + [\![XFY]\!][\![XF]\!]^\mathsf{T}$ ▷ symmetric
11: **return** $\begin{pmatrix} [\![X + XFYF^\mathsf{T}X]\!] & -[\![XFY]\!] \\ -[\![XFY]\!]^\mathsf{T} & [\![Y]\!] \end{pmatrix}$

resulting matrix is symmetric, which is to be exploited in an implementation. It is easy to see that protocol ObliviousRGInverse is oblivious: it only branches on the dimensions of the matrix, which are considered public, and otherwise only performs elementary arithmetic operations, and calls to secure subprotocols (including recursive calls to itself).

1.3.1 Correctness Analysis

Rohde [97] shows that a reflexive generalized inverse A^- of a symmetric, positive-semidefinite matrix *over the real numbers*[2]

$$A = \begin{pmatrix} E & F \\ F^\mathsf{T} & G \end{pmatrix} \tag{1.9}$$

can be expressed in Banachiewicz–Schur form as

$$A^- = \begin{pmatrix} E^- + E^-FS^-F^\mathsf{T}E^- & -E^-FS^- \\ -S^-F^\mathsf{T}E^- & S^- \end{pmatrix}, \tag{1.10}$$

where E^- is a reflexive generalized inverse of E and S^- is a reflexive generalized inverse of $S = G - F^\mathsf{T}E^-F$. This form allows for a block-recursive algorithm for computing the reflexive generalized inverse over the real numbers. As proved by Marsaglia and Styan, the correctness of Rohde's result *over an arbitrary field* depends on the following additional condition.

Lemma 1 ([73], statement tailored to our needs)**.** *Over an arbitrary field, Equation* (1.10) *is a reflexive generalized inverse of A if and only if*

$$\operatorname{rank} A = \operatorname{rank} E + \operatorname{rank} S, \tag{1.11}$$

[2]Rohde [97] actually shows his result for complex matrices, but for our purposes it is more convenient to state his result for real matrices.

or, equivalently, the following three conditions are satisfied simultaneously

$$\begin{cases} (I-EE^-)F(I-S^-S)=0 & (1.12)\\ (I-SS^-)F^{\mathsf{T}}(I-E^-E)=0 & (1.13)\\ (I-EE^-)FS^-F^{\mathsf{T}}(I-E^-E)=0, & (1.14)\end{cases}$$

where E^- and S^- are reflexive generalized inverses of E and $S=A/E$ respectively.

Lemma 2. *Over an arbitrary field, a sufficient condition for Equation* (1.10) *to be a reflexive generalized inverse of a symmetric matrix A is that A has generic rank profile.*

Proof. We partition A as in Equation (1.9) arbitrarily but such that E is square. Now we can make a case distinction on E: (i) E is invertible. Then E^- coincides with the ordinary inverse and it immediately follows that $(I-EE^-)=(I-E^-E)=0$, thus satisfying (1.12)–(1.14) from Lemma 1.

(ii) E is not invertible. Since A has generic rank profile, it then immediately follows that $\operatorname{rank}A=\operatorname{rank}E$ and furthermore that $\operatorname{rank}S=0$, thus satisfying (1.11). □ □

Lemma 3. *For any $m\times n$ matrix A over an arbitrary field, any k such that $A_{[k]}$ is invertible, and any i such that $0\le i\le k-\operatorname{rank}A$ it holds that*

$$A_{[k+i]}/A_{[k]}=(A/A_{[k]})_{[i]}.$$

Proof. Let

$$A=\begin{pmatrix}A_{11}&A_{12}&A_{13}\\A_{21}&A_{22}&A_{23}\\A_{31}&A_{32}&A_{33}\end{pmatrix},$$

where $A_{11}=A_{[k]}$ is an invertible $k\times k$ matrix and A_{22} is an $i\times i$ matrix. Then

$$\begin{aligned}(A/A_{[k]})_{[i]}&=\left(\begin{pmatrix}A_{22}&A_{23}\\A_{32}&A_{33}\end{pmatrix}-\begin{pmatrix}A_{21}\\A_{31}\end{pmatrix}A_{11}^{-1}\begin{pmatrix}A_{12}&A_{13}\end{pmatrix}\right)_{[i]}\\&=A_{22}-A_{21}A_{11}^{-1}A_{12}\\&=A_{[k+i]}/A_{[k]}.\end{aligned}$$

□

□

Corollary 4. *Protocol* ObliviousRGInverse*, when run on a symmetric matrix A over $\mathbb{F}_p$ having generic rank profile, correctly computes a reflexive generalized inverse.*

Proof. For the base case, we have already argued correctness of the extended reciprocal near the beginning of Section 1.3. For the recursive step applied to A, note that for an arbitrary partitioning but such that E is a $k\times k$ matrix for some integer k, it is easy to see that E is symmetric and has generic rank profile. Correctness then follows from Lemma 2.

We prove that S is symmetric and has generic rank profile by distinguishing two cases. If E is not invertible, then $\operatorname{rank}A=\operatorname{rank}E$ and S is necessarily the (square) null matrix, which is symmetric and has generic rank profile. Otherwise, E is invertible and $S=A/E=G-F^TE^{-1}F$, which is clearly symmetric. For generic rank profile, we can apply Schur's determinant formula to the leading principal minors of A: for any i such that $0\le i\le\operatorname{rank}A-k$ we have $0\neq\det(A_{[k+i]})=\det(E)\det(A_{[k+i]}/E)$. Then, applying Lemma 3 gives $\det(A_{[k+i]}/E)=\det((A/E)_{[i]})\neq 0$, i.e., A/E has generic rank profile. In both cases, correctness now follows from Lemma 2. □ □

Remark. We have proved that generic rank profile is sufficient for correctness—we did not prove that this condition is necessary. This leaves open the possibility that a weaker condition on the input matrix (weaker than generic rank profile) would suffice for correctness of ObliviousRGInverse. In the next section we will compute $(AA^{\mathsf{T}}AA^{\mathsf{T}})^{-}$, from which we construct $A^{\dagger}$. To ensure the correctness of ObliviousRGInverse we will actually randomize its input, $AA^{\mathsf{T}}AA^{\mathsf{T}}$, so that it has generic rank profile with high probability and then undo the randomization on the result. One might raise the question whether choosing the modulus p large enough to guarantee the existence of $A^{\dagger}$, could immediately guarantee correctness of ObliviousRGInverse without requiring $AA^{\mathsf{T}}AA^{\mathsf{T}}$ to have generic rank profile. We do not address this question, as our randomization technique suffices and introduces only minimal overhead.

1.3.2 Complexity Analysis

We first state the complexity (number of secure operations) of protocol Oblivious-RGInverse when run on a square matrix whose dimensions are a power of two.

Proposition 5. *Protocol* ObliviousRGInverse*, when run on an $m \times m$ matrix over $\mathbb{F}_p$, where $m = 2^k$ for integer k, requires $\frac{3}{2}m(m-1) + \frac{1}{2}m\log_2 m$ secure inner products and m invocations of* ScalarRGInverse*.*

Proof. Correctness of Proposition 5 is easily proved using induction on k. In the base case, $k = 0$, protocol ObliviousRGInverse simply invokes ScalarRGInverse once.

As induction hypothesis, suppose the proposition holds for some $m = 2^k$, where k is integer. Then protocol ObliviousRGInverse, when run on a $2m \times 2m$ matrix over $\mathbb{F}_p$, performs 2 invocations of OblivousRGInverse on $m \times m$ matrices, which requires $3m(m-1) + m\log_2 m$ secure inner products and $2m$ invocations of ScalarRGInverse per the induction hypothesis. The protocol further performs four matrix-matrix products of $m \times m$ matrices. Two of these products are symmetric, so these products can be performed using $3m^2 + m$ secure inner products. The total number of secure inner products required is therefore equal to $6m^2 - 2m + m\log_2 m = 6m^2 - 3m + m\log_2(2m) = \frac{3}{2}(2m)(2m-1) + \frac{1}{2}(2m)\log_2(2m)$ and the number of invocations of ScalarRGInverse is $2m$. □ □

If the dimensions of the matrix, m, are not a power of two, it is not always possible to divide the matrix evenly in step 4 of the protocol. In these cases the number of secure inner products required is slightly greater than the number stated in Proposition 5. For general dimensions, we prove the following proposition. We note that this bound is not tight.

Proposition 6. *Protocol* ObliviousRGInverse*, when run on an $m \times m$ matrix over $\mathbb{F}_p$, requires fewer than $\frac{3}{2}m(m-1) + m\log_2 m$ secure inner products and exactly m invocations of* ScalarRGInverse*.*

We also express the complexity of protocol ObliviousRGInverse in terms of elementary secure multiplications, for MPC schemes for which the "cheap inner product" is not available. Note that the bound given here is exact if we assume the naïve algorithm for matrix multiplication. A more advanced algorithm would result in sub-cubic, but still super-quadratic complexity.

Proposition 7. *Protocol* ObliviousRGInverse*, when run on an $m \times m$ matrix over $\mathbb{F}_p$, requires at most $\frac{1}{2}m^3 + \frac{1}{2}m^2 - m$ secure multiplications and exactly m invocations of* ScalarRGInverse*.*

The proofs of Proposition 6 and 7 can be found in the appendix.

1.4 Computing the Moore–Penrose Pseudoinverse

We will compute the Moore–Penrose pseudoinverse using a formula (see, e.g., [93, p. 207]) that computes $A^\dagger$ in terms of a reflexive generalized inverse:

$$A^\dagger = A^\mathsf{T}(AA^\mathsf{T}AA^\mathsf{T})^- AA^\mathsf{T}. \tag{1.15}$$

Before proposing our protocol Pseudoinverse, we deal with three remaining questions, namely how to compute the common denominator, how to choose an appropriate modulus, and how to reliably compute $(AA^\mathsf{T}AA^\mathsf{T})^-$, as $AA^\mathsf{T}AA^\mathsf{T}$ does not necessarily have generic rank profile, which is required by protocol ObliviousRGInverse for correctness.

1.4.1 Computing the Common Denominator

Over the rational numbers, a common denominator d such that $dA^\dagger$ is integer-valued if A is integer-valued is $d = (\operatorname{vol} A)^2$ [99, Satz 10]. The squared volume is minimal in the sense that there exist matrices for which it is the smallest possible common denominator.

If we would have an *orthonormal* basis for the left or right nullspace of A, then we could use [10, Thm. (4.1)] to compute $(\operatorname{vol} A)^2$ directly. An orthonormal basis does not necessarily exist over an arbitrary field. Instead, we generalize [10, Thm. (4.1)] by relaxing the requirements on the nullspace basis.

Lemma 8. *Let $A \in \mathbb{F}^{m\times k}$ be a matrix of rank r. Let $B \in \mathbb{F}^{m\times \ell}$ be a matrix of rank $m-r$ such that its columns are orthogonal to the columns of A, i.e., $B^\mathsf{T}A = 0$. Then,*

$$\det(AA^\mathsf{T} + BB^\mathsf{T}) = (\operatorname{vol} A)^2(\operatorname{vol} B)^2.$$

Proof. Note that $AA^\mathsf{T} + BB^\mathsf{T} = \begin{pmatrix} A & B \end{pmatrix}\begin{pmatrix} A & B \end{pmatrix}^\mathsf{T}$. Because the columns of A are orthogonal to those of B, the matrix $\begin{pmatrix} A & B \end{pmatrix}$ has rank $r + (m-r) = m$ and hence

$$\det(\begin{pmatrix} A & B \end{pmatrix}\begin{pmatrix} A & B \end{pmatrix}^\mathsf{T}) = (\operatorname{vol}\begin{pmatrix} A & B \end{pmatrix})^2 = (\operatorname{vol} A)^2(\operatorname{vol} B)^2,$$

where the first equality holds by equation (1.8), and the second equality is [10, Example 5.1]. □ □

Theorem 9. *Let $A \in \mathbb{F}^{m\times n}$ be a matrix of rank r. Let $K = I - AA^\dagger \in \mathbb{F}^{m\times m}$. Then,*

$$(\operatorname{vol} A)^2 = \det(AA^\mathsf{T} + K).$$

Proof. By property (3) of the pseudoinverse, we have that $K = K^\mathsf{T}$. This fact, and property (1) of the pseudoinverse imply that $KK^\mathsf{T} = KK = K$ and $K^\mathsf{T}A = 0$, i.e., K is idempotent and its columns are orthogonal to the columns of A.

Combining equation (1.6) with the fact that K is idempotent and symmetric gives us that $\operatorname{vol} K = \operatorname{vol}(KK^\mathsf{T}) = (\operatorname{vol} K)^2$. Since the volume of a matrix is nonzero, we conclude that $\operatorname{vol} K = 1$.

Orthogonality of the columns of K and A implies that $\operatorname{rank} K \le m - r$ and

$$\operatorname{rank} K = \operatorname{rank}(I - AA^\mathsf{T}) \ge \operatorname{rank} I - \operatorname{rank}(AA^\mathsf{T}) = m - r$$

follows from subadditivity of matrix rank. Applying Lemma 8 gives us

$$\det(AA^\mathsf{T} + K) = \det(AA^\mathsf{T} + KK^\mathsf{T}) = (\operatorname{vol} A)^2(\operatorname{vol} K)^2 = (\operatorname{vol} A)^2. \qquad \square$$

□

1.4.2 Bound on the Modulus

Springer [99] has proved the following upper bound on the magnitudes of the numerators and the common denominator of the pseudoinverse. Choosing p larger than twice this bound will guarantee that: (i) $d = (\mathrm{vol}A)^2$ is an invertible element in $\mathbb{F}_p$, which is a necessary and sufficient condition for existence of $A^\dagger$ over $\mathbb{F}_p$ [6] (see also Section 1.2), and (ii) that the pair $(dA^\dagger, d)$ over $\mathbb{F}_p$ coincides with $(dA^\dagger, d)$ over $\mathbb{Z}$ (see Lemma 10 below), and (iii) that the product $AA^\mathsf{T}AA^\mathsf{T}$ occurring in Equation (1.15) has the same rank as A (which we will need in Theorem 14, and note that (iii) is implied by applying (i) to the upcoming Proposition 11).

Lemma 10 ([99, Satz 12])**.** *Let $N_0 = (\mathrm{vol}A)^2$ and $Z_0 = (z_{ij}) \in \mathbb{Z}^{m\times n}$ be an integer matrix of rank r such that $A^\dagger = \frac{1}{N_0}Z_0$. Let $\mu = \min(m,n)$. Then,*

$$\max(|N_0|, \max_{i,j}|z_{ij}|) \leq \max\left(\frac{\|A\|_F^{2r}}{r^r}, \frac{\|A\|_F^{2r-1}}{\sqrt{r^r(r-1)^{r-1}}}\right), \tag{1.16}$$

and

$$\max(|N_0|, \max_{i,j}|z_{ij}|) \leq \max\left(\frac{\|A\|_F^{2\mu}}{\mu^\mu}, \frac{\|A\|_F^{2\mu-1}}{\sqrt{\mu^\mu(\mu-1)^{\mu-1}}}\right), \tag{1.17}$$

where $\|A\|_F = \sqrt{\sum_{ij}|a_{ij}|^2}$ is the Frobenius norm of A.

Remark. In a setting in which the rank r is unknown, one would use (1.17).

For our construction, we further require that

$$\mathrm{rank}(AA^\mathsf{T}AA^\mathsf{T}) = \mathrm{rank}A. \tag{1.18}$$

This requirement holds unconditionally over fields of characteristic zero, but not necessarily over finite fields. Nonetheless, as we show below, it turns out that existence of the Moore–Penrose inverse already implies (1.18).

Proposition 11. *Let A be an arbitrary matrix over $\mathbb{F}$. The Moore–Penrose inverse of A exists if and only if*

$$\mathrm{rank}(AA^\mathsf{T}AA^\mathsf{T}) = \mathrm{rank}A.$$

Proof. Recall from Section 1.2 that the Moore–Penrose inverse exists over $\mathbb{F}$ if and only if $\mathrm{rank}(AA^\mathsf{T}) = \mathrm{rank}(A^\mathsf{T}A) = \mathrm{rank}A$. Let $A = VW$ be a rank decomposition of A, i.e., V and W have full column-rank and full row-rank, respectively. Over an arbitrary field, a rank decomposition exists but is not necessarily unique; see, e.g., [92]. Then,

$$\mathrm{rank}(AA^\mathsf{T}) = \mathrm{rank}(VWW^\mathsf{T}V^\mathsf{T}) = \mathrm{rank}A \iff \mathrm{rank}WW^\mathsf{T} = \mathrm{rank}A,$$

and similarly,

$$\mathrm{rank}(A^\mathsf{T}A) = \mathrm{rank}(W^\mathsf{T}V^\mathsf{T}VW) = \mathrm{rank}A \iff \mathrm{rank}V^\mathsf{T}V = \mathrm{rank}A.$$

Also note that both WW^T and $V^\mathsf{T}V$ have dimension $r \times r$ with $r = \mathrm{rank}A$, i.e., they are invertible. We now write $AA^\mathsf{T}AA^\mathsf{T}$ in terms of V and W, and multiply by V^T from the left and by V from the right, by which we obtain :

$$V^\mathsf{T}AA^\mathsf{T}AA^\mathsf{T}V = (V^\mathsf{T}V)(WW^\mathsf{T})(V^\mathsf{T}V)(WW^\mathsf{T})(V^\mathsf{T}V).$$

Thus, $\mathrm{rank}V^\mathsf{T}AA^\mathsf{T}AA^\mathsf{T}V = \mathrm{rank}A$, if and only if $\mathrm{rank}AA^\mathsf{T}AA^\mathsf{T} = \mathrm{rank}A$. □ □

1.4.3 Symmetric Preconditioning

A *preconditioner* is a mapping $A \mapsto h(A)$ for matrices A from a given class, where the goal is to achieve a certain property, either with certainty or with high probability. This property is typically an input condition from some computational technique. For a more elaborate and formal introduction into preconditioning we refer to [20]. Here, we restrict to preconditioners for achieving *generic rank profile* for symmetric matrices of the form $A = BB^{\mathsf{T}}$ over an arbitrary field of positive characteristic.

To ensure correctness of protocol ObliviousRGInverse, we need a preconditioner with the following three properties:

(i) achieves generic rank profile with high probability;

(ii) preserves symmetry, i.e., $h(A)$ is symmetric;

(iii) is *removable*. Informally speaking, this means that the preconditioner can be efficiently removed once "it has done its job". Formally, a preconditioner is removable with respect to computing a reflexive generalized inverse if there exists an efficiently computable mapping g such that $g(h(A)^-) \in \mathscr{A}^-$, where $\mathscr{A}^-$ denotes the set of reflexive generalized inverses of A.

Although several preconditioners for achieving generic rank profile have been proposed in the literature, we are not aware of an existing result that covers all of the above properties simultaneously. For example, the Toeplitz preconditioner by Kaltofen and Saunders [60] fails to satisfy (ii), and the diagonal preconditioner proposed in [39] (combined with a suitable linear-independence preconditioner, see [20]) fails to satisfy (iii).

In this section we will show that for a symmetric matrix A, the preconditioner $h(A) = UAU^{\mathsf{T}}$ with U a uniformly random (invertible) matrix is sufficient for satisfying (i)–(iii). It is easy to see that (ii) holds. We prove property (i) in Theorem 14 and (iii) in Lemma 15.

Lemma 12 (Schwartz–Zippel)**.** *Let $g \in \mathbb{F}[x_1, \ldots, x_n]$ be a nonzero polynomial of total degree $d \geq 0$ over a field $\mathbb{F}$. Let $\mathscr{S} \subseteq \mathbb{F}$ and let $\alpha_1, \ldots, \alpha_n$ be chosen independently and uniformly at random from $\mathscr{S}$. Then,*

$$\Pr[g(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|\mathscr{S}|}.$$

Lemma 13 (See, e.g., [16, Lem. 2-(iii)])**.** *The probability that a uniformly random matrix $U \in \mathbb{F}^{m \times m}$ is invertible equals*

$$\Pr(\det U \neq 0) = \prod_{k=1}^{m} \left(1 - |\mathbb{F}|^{-k}\right).$$

Theorem 14. *Let $A \in \mathbb{F}^{m \times n}$ be arbitrary, let r be the rank of A and let AA^{T} have the same rank as A. Let $U \in \mathbb{F}^{m \times m}$ be chosen uniformly at random. Then, the probability that U is invertible and $UAA^{\mathsf{T}}U^{\mathsf{T}}$ has generic rank profile is*

$$\Pr_U \left(\det U \neq 0 \wedge [UAA^{\mathsf{T}}U^{\mathsf{T}}]_k \neq 0 \quad \forall k \in [r]\right) > 1 - \frac{r(r+1)+2}{|\mathbb{F}|}.$$

Proof. We view $U = (u_{i,j})$ as a polynomial matrix with $u_{i,j}$ as indeterminates. For every $1 \leq k \leq r$, we apply the Cauchy–Binet formula to obtain an expression for the leading principal minor of order k of the matrix $UAA^{\mathsf{T}}U^{\mathsf{T}}$, which is a polynomial in the variables $u_{i,j}$, where we let $\mathscr{K} = [k]$,

$$\begin{aligned} f_k(u_{1,1},\ldots,u_{i,j},\ldots,u_{m,m}) &= [UAA^{\mathsf{T}}U^{\mathsf{T}}]_{\mathscr{K},\mathscr{K}} \\ &= \sum_{\substack{\mathscr{I}\subset[m]\\|\mathscr{I}|=k}} [UA]_{\mathscr{K},\mathscr{I}}[A^{\mathsf{T}}U^{\mathsf{T}}]_{\mathscr{I},\mathscr{K}} = \sum_{\substack{\mathscr{I}\subset[m]\\|\mathscr{I}|=k}} \left([UA]_{\mathscr{K},\mathscr{I}}\right)^2 \\ &= \sum_{\substack{\mathscr{I}\subset[m]\\|\mathscr{I}|=k}} \Big(\sum_{\substack{\mathscr{J}\subset[m]\\|\mathscr{J}|=k}} [U]_{\mathscr{K},\mathscr{J}}[A]_{\mathscr{J},\mathscr{I}}\Big)^2. \end{aligned}$$

It follows immediately from the structure of this formula that the total degree of f_k is $2k$.

Let us now prove that none of the polynomials f_k for all $1 \leq k \leq r$ is equal to the zero polynomial. Because AA^{T} is symmetric, there exists an invertible matrix $S = (s_{i,j})$ such that $SAA^{\mathsf{T}}S^{\mathsf{T}} = \Lambda$ where $\Lambda = \operatorname{diag}(\lambda_1,\ldots,\lambda_r,0,\ldots,0)$ with $\lambda_i \neq 0$ for all $1 \leq i \leq r$ [1, Thm. 6]. Hence,

$$f_k(s_{1,1},\ldots,s_{i,j},\ldots,s_{m,m}) = \prod_{i=1}^{k} \lambda_i \neq 0 \qquad \forall k \in [r].$$

The Schwartz–Zippel lemma asserts that $\Pr[f_k(U_{1,1},\ldots,U_{m,m}) = 0] \leq \frac{2k}{|\mathbb{F}|}$, where the $U_{i,j}$ represent the elements of U when viewed as (uniformly random and independent) random variables. Hence, by applying the union bound over k we obtain

$$\Pr[f_1(U) \neq 0 \wedge \cdots \wedge f_r(U) \neq 0] \geq 1 - \frac{\sum_{k=1}^{r} 2k}{|\mathbb{F}|} = 1 - \frac{r(r+1)}{|\mathbb{F}|}.$$

Combining this bound with that of Lemma 13 gives

$$\begin{aligned} &\Pr_U(\det U \neq 0 \wedge [UAA^{\mathsf{T}}U^{\mathsf{T}}]_k \neq 0 \quad \forall k \in [r]) \\ &\quad \geq \prod_{k=1}^{m} \left(1 - |\mathbb{F}|^{-k}\right) - \frac{r(r+1)}{|\mathbb{F}|} > \frac{|\mathbb{F}|-2}{|\mathbb{F}|-1} - \frac{r(r+1)}{|\mathbb{F}|} > 1 - \frac{r(r+1)+2}{|\mathbb{F}|}, \end{aligned}$$

where we used that

$$\begin{aligned} \prod_{k=1}^{m}(1-x_k) > \prod_{k=1}^{\infty}(1-x_k) &= 1 - x_1 - x_2(1-x_1) - x_3(1-x_1)(1-x_2) - \ldots \\ &> 1 - \sum_{k=1}^{\infty} x_k. \end{aligned}$$

With $x_k = |\mathbb{F}|^{-k}$, we get that $\prod_{k=1}^{m}\left(1-|\mathbb{F}|^{-k}\right) > 1 - (|\mathbb{F}|-1)^{-1}$. □ □

We now prove that the preconditioner $h(A) = UAU^{\mathsf{T}}$ with invertible U is removable.

Lemma 15. *Let A be a matrix over $\mathbb{F}$ and let $\mathscr{A}^-$ denote the set of reflexive generalized inverses of A. Let U be an invertible matrix over $\mathbb{F}$ and let $Y = (UAU^{\mathsf{T}})^-$ be a reflexive generalized inverse of UAU^{T}. Then, $U^{\mathsf{T}}YU \in \mathscr{A}^-$.*

Proof. Given the Penrose equations (1) and (2) for Y, we need to show that the Penrose equations (1) and (2) hold for A^-. Since U is invertible,

$$A(U^\mathsf{T}YU)A = U^{-1}(UAU^\mathsf{T})Y(UAU^\mathsf{T})(U^\mathsf{T})^{-1} = U^{-1}(UAU^\mathsf{T})(U^\mathsf{T})^{-1} = A.$$

Furthermore,

$$(U^\mathsf{T}YU)A(U^\mathsf{T}YU) = U^\mathsf{T}Y(UAU^\mathsf{T})YU = U^\mathsf{T}YU. \qquad \square$$

$\square$

1.4.4 Construction

Our protocol Pseudoinverse, on input of a secret-shared matrix $[\![A]\!] \in \mathbb{F}_p^{m\times n}$, computes the pair $([\![A^\dagger]\!], [\![(\mathrm{vol}A)^2]\!])$ and is given as Protocol 3. Protocol Pseudoinverse makes use of a secure subprotocol Determinant for computing the determinant of an invertible matrix in $\mathbb{F}_p^{m\times m}$ in secret-shared form. A possible instantiation of Determinant can be found in [23], where it is called protocol $\blacksquare_0$. See also [13], which slightly modifies this protocol to reduce its randomness complexity.

Protocol 3 Pseudoinverse($[\![A]\!]$)

1: **if** $m > n$ **then**
2: **return** Pseudoinverse($[\![A]\!]^\mathsf{T}$)$^\mathsf{T}$
3: $[\![AA^\mathsf{T}]\!] \leftarrow [\![A]\!][\![A]\!]^\mathsf{T}$ ▷ symmetric
4: $[\![AA^\mathsf{T}AA^\mathsf{T}]\!] \leftarrow [\![AA^\mathsf{T}]\!][\![AA^\mathsf{T}]\!]$ ▷ symmetric
5: $U \xleftarrow{\$} \mathbb{F}_p^{m\times m}$
6: $[\![X]\!] \leftarrow U^\mathsf{T}$ObliviousRGInverse$(U[\![AA^\mathsf{T}AA^\mathsf{T}]\!]U^\mathsf{T})U$
7: $[\![XAA^\mathsf{T}]\!] \leftarrow [\![X]\!][\![AA^\mathsf{T}]\!]$
8: $[\![A^\dagger]\!] \leftarrow [\![A^\mathsf{T}]\!][\![XAA^\mathsf{T}]\!]$
9: $[\![K]\!] \leftarrow I - [\![AA^\mathsf{T}]\!][\![XAA^\mathsf{T}]\!]$ ▷ symmetric; in parallel with $[\![A^\dagger]\!]$
10: $[\![d]\!] \leftarrow$ Determinant$([\![AA^\mathsf{T}]\!] + [\![K]\!])$
11: **return** $([\![A^\dagger]\!], [\![d]\!])$

We note that the rank of A is given by $\mathrm{Tr}(AA^\dagger)$ [17]. It can be computed obliviously in Pseudoinverse as $[\![r]\!] = m - \mathrm{Tr}([\![K]\!])$.

Corollary 16. *Protocol* Pseudoinverse*, when run on an arbitrary $m \times n$ matrix over $\mathbb{F}_p$, correctly computes the Moore–Penrose pseudoinverse with probability at least*

$$\Pr(\mathit{success}) \geq \left[1 - \frac{m(m+1)+2}{|\mathbb{F}|}\right] \cdot P_{\mathsf{Determinant}},$$

where $P_{\mathsf{Determinant}}$ denotes the success probability of protocol Determinant*.*

1.4.5 Complexity Analysis

Proposition 17. *Protocol* Pseudoinverse*, when run on an arbitrary $m \times n$ matrix over $\mathbb{F}_p$, requires $mn + \frac{5}{2}m^2 + \frac{3}{2}m$ secure inner products (or: $m^2n + \frac{5}{2}m^3 + \frac{3}{2}m^2$ secure multiplications), one invocation of protocol* Determinant *on a symmetric $m \times m$ matrix and one invocation of* ObliviousRGInverse *on a symmetric $m \times m$ matrix.*

Protocol Determinant, instantiated as in [13], when invoked on a $m \times m$ matrix, requires secure sampling of m^2 random elements, and performing $2m^2+m-1$ secure inner products (or: $\frac{4}{3}m^3+\frac{2}{3}m-1$ secure multiplications) and m^2 open operations.

The field inversion technique from Bar-Ilan and Beaver [7] requires secure sampling of one random element and one secure multiply-and-open operation.

Subprotocol IsZero can be instantiated with the probabilistic secure zero test from Nishide and Ohta [83]. This secure zero test is constant round and requires 2κ secure multiplications, 4κ secure multiply-and-open operations and secure sampling of 5κ random elements, where κ is a security parameter and the protocol may fail with probability $2^{-\kappa}+1/p$.

Corollary 18. *Protocol* Pseudoinverse*, when run on an arbitrary $m \times n$ matrix over $\mathbb{F}_p$, with protocol* Determinant *instantiated as in [13], requires in total $nm+6m^2+o(m^2)$ secure inner products (or: $nm^2+\frac{13}{3}m^3+o(m^3)$ secure multiplications), m^2 public random elements, m^2 private random elements, m^2 openings, m secure zero tests and m secure field inversions.*

If protocols IsZero *and* Reciprocal *are instantiated as the probabilistic zero test from [83] and as in [7], respectively, the m secure zero tests and field inversions require $O(\kappa m)$ secure multiplications, random elements and openings.*

Remark. It is straightforward to adapt Protocol 3 such that in line 8 it computes the vector $A^\dagger b$ instead of the matrix $A^\dagger$, i.e., directly solving the linear system $Ax=b$ for the vector x. By replacing line 8, in case $m \leq n$, with the two lines $[\![XAA^\mathsf{T}b]\!] \leftarrow [\![XAA^\mathsf{T}]\!][\![b]\!]$ and $[\![A^\dagger b]\!] \leftarrow [\![A^\mathsf{T}]\!][\![XAA^\mathsf{T}b]\!]$, one can avoid the matrix-matrix product that gives rise to the mn term. Namely, the complexity (number of secure inner products) becomes $O(n+m^2)$. If $m > n$ we would transpose the system to be solved: $x^\mathsf{T}A^\mathsf{T} = b^\mathsf{T}$. In this case, line 8 would be replaced by two secure products in which the matrix is multiplied from the *left* by the vector and this would result in a complexity of $O(n^2)$ secure inner products. Note, however, that this adaptation imposes an additional constraint on the size of the modulus; the field should now be large enough to uniquely represent the coefficients of the vector $dA^\dagger b$.

1.5 Proofs of Proposition 6 and 7

The proof is by complete induction. It is clear that the number of invocations of ScalarRGInverse required when ObliviousRGInverse is run on an $m \times m$ matrix is equal to m. Let $D(m)$ denote the number of secure inner products required to run protocol ObliviousRGInverse on an $m \times m$ matrix. Similarly, let $M(m)$ be the number of secure multiplications required, in case a "cheap inner product" is not available. Then, we have to show that

$$D(m) < \frac{3}{2}m(m-1)+m\log_2 m; \qquad \text{and} \tag{1.19}$$

$$M(m) \leq \frac{1}{2}m^3+\frac{1}{2}m^2-m. \tag{1.20}$$

Inspection of the protocol shows that

$$D(1) = 0;$$

$$D(2k) = 2D(k)+3k^2+k; \text{ and} \tag{1.21}$$

$$D(2k+1) = D(k)+D(k+1)+3k^2+4k+1, \tag{1.22}$$

where we distinguish between even ($m = 2k$) and odd ($m = 2k+1$) dimensions. Similarly, for $M(m)$, we have

$$M(1) = 0;$$
$$M(2k) \leq 2M(k) + 3k^3 + k^2; \text{ and} \tag{1.23}$$
$$M(2k+1) \leq M(k) + M(k+1) + 3k^3 + \frac{11}{2}k^2 + \frac{5}{2}k. \tag{1.24}$$

The inequalities in (1.23) and (1.24) can be replaced with equalities in case the naïve algorithm for matrix multiplication is used.

In the base case $m = 1$, the propositions clearly hold. Assume the propositions hold for all $m' < m$. Then for odd $m = 2k+1$ substitution of (1.19) in (1.22) yields

$$\begin{aligned} D(2k+1) &< \frac{3}{2}m(m-1) + k + 1 + k\log_2 k + (k+1)\log_2(k+1) \\ &= \frac{3}{2}m(m-1) + \log_2\left(k^k(k+1)^{k+1}2^{k+1}\right) \\ &< \frac{3}{2}m(m-1) + (2k+1)\log_2(2k+1), \end{aligned}$$

where the last inequality follows from monotonicity of the logarithm and from the following inequality:

$$\begin{aligned} 2^{k+1}k^k(k+1)^{k+1} &= 2^{1-k}(2k+1-1)^k(2k+1+1)^k(k+1) \\ &= 2^{1-k}((2k+1)^2-1)^k(k+1) \\ &< 2^{1-k}(2k+1)^{2k}(k+1) \\ &< (2k+1)^{2k+1}. \end{aligned}$$

For the case of even m, Proposition 6 follows immediately by substituting equation (1.19) in (1.21). Similarly, Proposition 7 follows from the substitution of (1.20) in (1.23) and (1.24).

Chapter 2

Secure Streaming Algorithms for Sums of Frequency Moments

2.1 Streaming algorithms

A *data stream* is an ordered sequence which can only be accessed sequentially. Now consider one desires the approximation of some function f with a data stream σ as input. A data stream potentially allows multi-pass data traversal, in this case the stream should obviously be of finite length. Whenever the data stream is of finite length we will denote this length with m. Formally, let $\sigma = (x_1, \ldots, x_m)$ denote a stream of finite length m, where each stream element $x_i \in [n]$. The main goal is to process the stream in data space much less than $m \log n$, preferably sub-linear or even logarithmic in both n and m.

The streaming model of computation dates back to papers by Munro and Paterson [80] and Morris [78]. The original motivation to study the streaming model of computation stems from limited hardware capabilities such as a limited amount of memory. A randomized *streaming algorithm* $\mathscr{A}$ for function f takes the stream σ as input and keeps a summary s, called the *sketch*. On processing a stream item x_i, the streaming algorithm updates the sketch. The streaming algorithm uses the sketch to approximate the true answer $f(x_1, x_2, \ldots)$. The quality of the sketch is defined as follows.

Definition 19. Let the random variable $\mathscr{A}(\sigma)$ denote the output provided by streaming algorithm $\mathscr{A}$ on input $\sigma = (x_1, x_2, \ldots)$. We say that algorithm $\mathscr{A}$ (ε, δ)-*approximates* f when

$$\Pr\left[\left|\frac{\mathscr{A}(\sigma)}{f(\sigma)} - 1\right| > \varepsilon\right] < \delta$$

We shall focus on (ε, δ) sum of the kth order frequency moments:

$$F_k := \sum_{i=1}^{n} f_i^k, \text{ where } f_i \text{ is the frequency of item } i.$$

Note that the 0th order sum equals the number of *distinct elements n* while the 1st order sum equals the total number of *events m*. Higher orders contains additional interesting statistical properties of the data, e.g., the 2nd order sum is known as *Simpson's index* in ecology, the *Herfindall-Hirschman index* in economics and is the main component of the *Gini heterogeneity index*.

2.2 Distinct elements

Counting the number of distinct elements in a stream σ has been researched extensively [42, 2, 63, 8, 38, 40, 12] In its generalized form the distinct element count equals F_0, the sum of the zeroth order moments of the stream's frequencies f_i.

We study one such algorithm originally proposed by [42] and improved by Alon, Matias and Szegedy [2]. As a building block, we define a function $\rho_b : \{0,1\}^\ell \to \mathbb{N}_{\leq \ell}$ which returns the number of least significant zeros or ones for $b = 0$ and $b = 1$ respectively:

$$\rho_b(x) = \begin{cases} \ell & \text{if } x = b(2^\ell - 1), \\ \max\{i \,:\, (2^i - b) | x\} & \text{otherwise.} \end{cases}$$

The role of ρ_0 in Algorithm 4 is to take hashed input values and assign them a 'score' with a known fixed distribution. This 'score' is defined as the number of consecutive zeros. Sketch s then stores the maximum 'score' observed. Alon et al. observe that s is a good estimator for $\log_2(F_0)$, because the expected number of times of the event $\rho_0(h(x_i)) \geq s$, for given s, is $F_0/2^s$.

Protocol 4 AMS F_0 [2]

Let ℓ be the bit-length of n.

Sketch(σ,ℓ):

- $a, b \in_R \mathbb{F}_{2^\ell}$ ▷ $h(x) = ax + b$
- $s \leftarrow 0$
- **Process** x **from** σ
  - $u \leftarrow \rho_0(ax + b)$
  - **if** $s < u$ **then** ▷ $s = \max \rho_0(h(x_i))$
    - $s \leftarrow u$
- **return** s

Output(s):

- **return** $\hat{F}_0 \leftarrow 2^s$

The plaintext AMS F_0 sketch, as shown in Algorithm 4, requires us to keep track of the maximum ρ_0 value observed from hashed incoming values. This is done by storing a current maximum s and for every incoming stream element x an update $u = \rho_0(h(x))$ is computed and compared against s. In secure multiparty computation such a comparison $(s < u)$ effectively forces us to bit-decompose s and u. In order to save one bit-decomposition per comparison we represent our sketch strategically as a bit-string $\bar{s}$. We let $\bar{s}$ be a unary representation of s such that $\rho_0(\bar{s}) = s$. Formally, we define $\bar{s} := (s_{\ell-1}, \ldots, s_0)$ with

$$s_i = \begin{cases} 0 & \text{if } i < s, \\ 1 & \text{otherwise.} \end{cases}$$

Note that indeed $\rho_0(\bar{s}) = s$ by design. Since we change representation of sketch s, we should change representation of update u accordingly. To this end the result of an incoming hashed stream value is bit-decomposed. We then use Prefix-OR to represent our updates $\bar{u}$, and perform the update as an AND operation. Since ρ_0 is invariant under Prefix-OR this update preserves the desired highest number of consecutive (least significant) zeros. See Figure 2.1 for a visual representation of an example update step where the pre-updated $s = 1$ is updated to $s' = 2$.

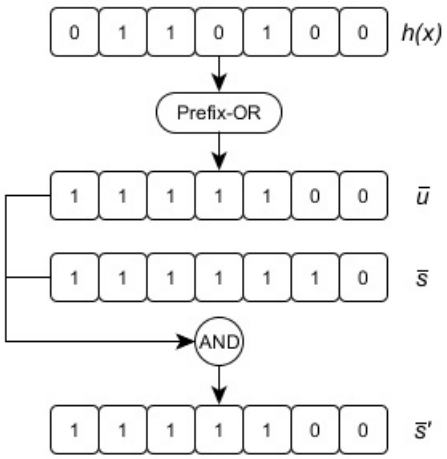


Figure 2.1: Secure F_0 sketch update example

Protocol 1 Secure F_0

Let ℓ be the bit-length of n.

Sketch(σ,ℓ):

$a, b \in_R \mathbb{F}_{2^\ell}$ ▷ $h([\![x]\!]) = a[\![x]\!] + b$

$[\![s_{\ell-1}]\!], \ldots, [\![s_0]\!] \leftarrow ([\![1]\!], \ldots, [\![1]\!])$ ▷ sketch $\bar{s} = (s_{\ell-1}, \ldots, s_0)$

Process $[\![x]\!]$ **from** σ

$[\![u]\!] \leftarrow a[\![x]\!] + b$

$[\![u_{\ell-1}]\!], \ldots, [\![u_0]\!] \leftarrow$ bit-decompose($[\![u]\!]$)

$[\![u_{\ell-1}]\!], \ldots, [\![u_0]\!] \leftarrow$ prefix-OR($[\![u_{\ell-1}]\!], \ldots, [\![u_0]\!]$)

$[\![s_{\ell-1}]\!], \ldots, [\![s_0]\!] \leftarrow$ AND($([\![u_{\ell-1}]\!], \ldots, [\![u_0]\!]), ([\![s_{\ell-1}]\!], \ldots, [\![s_0]\!])$)

return $[\![s_{\ell-1}]\!], \ldots, [\![s_0]\!]$

Output($[\![s_{\ell-1}]\!], \ldots, [\![s_0]\!]$):

$s_{\ell-1}, \ldots, s_0 \leftarrow$ open($[\![s_{\ell-1}]\!], \ldots, [\![s_0]\!]$)

$z \leftarrow \ell$

for $i \leftarrow 0$ **to** $\ell - 1$ **do**

if $s_i = 1$ **then**

$z \leftarrow i$

Break

return $F_0 \leftarrow 2^z$

Many streaming algorithm predictors, like the AMS F_0 sketch, do not achieve arbitrary precision.

Lemma 20. *[2, Proposition 2.3] The following probability bound holds for the estimator $\hat{F}_0$ obtained from Algorithm 4:*

$$\Pr\left[\frac{\hat{F}_0}{F_0} \notin \left[\frac{1}{c}, c\right]\right] \leq \frac{2}{c}, \quad \forall\, c > 2.$$

In general, the precision can be increased by running α instances of the protocol, preferably in parallel, and taking the median over all instances. This way this rather weak bound can be easily improved to a $(O(1), \delta)$ approximation, for $\delta > 0$ arbitrary, by running $\alpha = \Theta(1/\delta)$ instances.

Therefore, when designing secure protocols we require the sketch functionality to output a secret shared sketch $[\![s]\!]$. This way secret shared results of multiple protocol instance, over the same input stream, can be generated and used to produce the more accurate median result securely. Additionally, we retain the possibility to compute a plaintext output for a single sketch with the sketches native 'Output' functionality.

Protocol 2 Secure α-median F_0

for all $i \in [\alpha]$ **do in parallel**
  $[\![\bar{s}_i]\!] \leftarrow \text{Secure}F_0.\text{Sketch}(\sigma, \ell)$
Sort the $[\![\bar{s}_i]\!]$, by $\rho_0(\bar{s}_i)$, securely.
$v \leftarrow \lfloor (\alpha - 1)/2 \rfloor$ ▷ index starts at 0
$f_v \leftarrow \text{Secure}F_0.\text{Output}([\![\bar{s}_v]\!])$
if $k \equiv 0 \mod 2$ **then**
  $f_{v+1} \leftarrow \text{Secure}F_0.\text{Output}([\![\bar{s}_{v+1}]\!])$
  return $\frac{1}{2}(f_v + f_{v+1})$
return f_v

There are many different approaches to obliviously sort secret shared lists. Oblivious sorting is an interesting research topic in and of itself and beyond the scope of this paper.

Running $\text{Secure}F_0.\text{Output}([\![\bar{s}]\!])$ involves opening secret shared values. So, to guarantee security for Protocol 2, in the case of even k, we need to show that running Protocol 1's Output functionality for both s_v and s_{v+1} does not reveal information not already contained in Protocol 2's output $\frac{1}{2}(f_v + f_{v+1}) = \frac{1}{2}(2^z + 2^w)$. Thus we proof the following Lemma.

Lemma 21. *In the case of even k, revealing both $z = \rho_0(\bar{s}_v)$ and $w = \rho_0(\bar{s}_{v+1})$ does not reveal more than Protocol 2's output.*

Proof. When we observe a result $r = 2^{z-1} + 2^{w-1}$ we know that $w \geq z$ which allows us to rewrite $r = 2^{z-1}(1 + 2^{w-z})$, with $w - z \geq 0$. First, consider the case that $r \notin \mathbb{N}$. Because $1 + 2^{w-z} \in \mathbb{N}$ for any integers $w \geq z$, we must have $r = 2^{-1}(1 + 2^w)$ with $w \neq 0$, so we extract $z = 0$ and $w = \log_2(2r - 1)$. Second, for $r \in \mathbb{N}$, consider the case $\log_2(r) \in \mathbb{N}$. It follows that $z = w$, so we extract $z = w = \log_2(r)$. Third, for $r \in \mathbb{N}$, consider the case $\log_2(r) \notin \mathbb{N}$. We then extract $z = 1 + \max\{i : 2^i | r\}$ and $w = 1 + \log_2(r - 2^{z-1})$. □

2.3 Higher order frequency moments

Alon, Matias and Szegedy [2] also describe a streaming algorithm to estimate the sum of higher order frequency moments F_k for $k \geq 2$. Their approach involves keeping count of the number of occurrences of a 'representative' element a_i. The 'representative' element is replaces for a new element with probability $1/m$, where m is the current length of the stream. Note that k does not need to be decided upon in the Sketch phase but is merely required to compute the appropriate estimator $\hat{F}_k$ in the Output phase.

Protocol 5 AMS F_k $(k \geq 2)$ [2]

Sketch(σ,t):

for all $i \in [t]$ **do in parallel**
$s_i \leftarrow 0$ ▷ sketch s
$a_i \leftarrow Nil$
$m = 0$
Process x **from** σ
$m \leftarrow m+1$
for all $i \in [t]$ **do in parallel**
$b \leftarrow_{R(\frac{m-1}{m},\frac{1}{m})} \{0,1\}$
if $b = 1$ **then**
$a_i \leftarrow x$
$s_i \leftarrow 1$
else if $a_i = x$ **then**
$s_i \leftarrow s_i + 1$
return $(s_{t-1},\ldots,s_0),m$

Output$((s_{t-1},\ldots,s_0),m,k)$:

return $\hat{F}_k = \frac{1}{t}\sum_{i=1}^{t} m\left(s_i^k - (s_i-1)^k\right)$

Protocol 3 Secure F_k $(k \geq 2)$

Sketch(σ,t):

for all $i \in [t]$ **do in parallel**
$[\![s_i]\!] \leftarrow [\![1]\!]$ ▷ sketch s
$a_i \leftarrow None$
$m = 0$
Process $[\![x]\!]$ **from** σ
$m \leftarrow m+1$
for all $i \in [t]$ **do in parallel**
$[\![b]\!] \leftarrow_{R(\frac{m-1}{m},\frac{1}{m})} \{[\![0]\!],[\![1]\!]\}$
$[\![a_i]\!] \leftarrow [\![a_i]\!] + [\![b]\!]([\![x]\!] - [\![a_i]\!])$
$[\![\beta]\!] \leftarrow [\![(a_i = x)]\!]$
$[\![s_i]\!] \leftarrow [\![b]\!] + (1 - [\![b]\!])([\![s_i]\!] + [\![\beta]\!])$
return $([\![s_{t-1}]\!],\ldots,[\![s_0]\!]),m$

Output$(([\![s_{t-1}]\!],\ldots,[\![s_0]\!]),m,k)$:

$[\![Y]\!] \leftarrow [\![0]\!]$
for all $i \in [t]$ **do in parallel**
$[\![R]\!] \leftarrow (-1)^{k+2}[\![s_i]\!]$
$[\![Y]\!] \leftarrow (-1)^{k+1} + k[\![R]\!]$
for $j = 2$ **to** $k-1$ **do** ▷ Do not enter when $k = 2$
$[\![R]\!] \leftarrow -1 \cdot [\![s_i]\!] \cdot [\![R]\!]$
$[\![Y]\!] \leftarrow [\![Y]\!] + \binom{k}{j}[\![R]\!]$
return $[\![\hat{F}_k]\!] = \frac{m}{t}[\![Y]\!]$

Correctness of the output phase of Protocol 3 relies on the following equations:

$$s_i^k - (s_i - 1)^k = s_i^k - \sum_{j=0}^{k} \binom{k}{j} s_i^j (-1)^{k-j} = \sum_{j=0}^{k-1} \binom{k}{j} s_i^j (-1)^{k-j+1} = \sum_{j=0}^{k-1} \binom{k}{j} s_i^j (-1)^{k+1+j}$$

Now let $R^{\langle j \rangle}$ and $Y^{\langle j \rangle}$ represent the values of R and Y after the j'th iteration respectively. In protocol 3 R is defined as follows:

$$R^{\langle 0 \rangle} := (-1)^{k+1}, \quad R^{\langle j \rangle} := -1 \cdot s_i \cdot R^{\langle j-1 \rangle} \quad \forall j = 1, \ldots k-1$$

Which means

$$R^{\langle j \rangle} = s_i^j (-1)^{k+1+j}, \quad \forall j = 0, \ldots k-1. \tag{2.1}$$

Y is defined as follows:

$$Y^{\langle j \rangle} := \sum_{\alpha=0}^{j} \binom{k}{\alpha} R^{\langle \alpha \rangle}, \quad j = 0, \ldots, k-1$$

Which implies, together with Equation 2.1:

$$Y^{\langle k-1 \rangle} = \sum_{\alpha=0}^{k-1} \binom{k}{\alpha} s_i^\alpha (-1)^{k+1+\alpha} = s_i^k - (s_i - 1)^k$$

Just like with F_0 we can utilize the median trick to produce a (ε, δ)-approximation of F_k.

Theorem 22. *[2, Theorem 2.1] Let $t \geq 8kn^{1-1/k}/\delta^2$ and $\alpha \geq 2\log_2 1/\varepsilon$. Then the median of α runs of Algorithm 5 with parameter t provides a (ε, δ)-approximation of F_k*

Protocol 4 Secure α-median F_k

for all $i \in [\alpha]$ **do in parallel**
  $[\![\bar{s}_i]\!], m \leftarrow \text{Secure}F_k.\text{Sketch}(\sigma, t)$
  $[\![f_i]\!] \leftarrow \text{Secure}F_k.\text{Output}([\![\bar{s}_i]\!], m, k)$
Sort the $[\![f_i]\!]$ securely.
$v \leftarrow \lfloor (\alpha - 1)/2 \rfloor$ ▷ index starts at 0
if $k \equiv 0 \mod 2$ **then**
  return $\frac{1}{2}([\![f_v]\!] + [\![f_{v+1}]\!])$
return $[\![f_v]\!]$

Chapter 3

Efficient Private Map on Sparse Datasets & Kaplan-Meier Survival Analysis

This chapter presents an efficient private map operation and its application to Kaplan-Meier survival analysis. Specifically, the method optimizes the speed of the logrank test under MPC to decide if two Kaplan-Meier estimates are statistically different. This use case is introduced in SODA deliverable D4.1, will be elaborated on in section 3.5 below, and will be implemented SODA in deliverable D4.5.

3.1 Multi-Party Computation Framework

We propose a method for carrying out a "map" operation on a secret-shared dataset. We assume that the items in the datasets are vectors so that the full dataset is a matrix with the items as rows, secret-shared between a number of workers. (Either an inputter has secret-shared the dataset beforehand, or it was the result of a previous multiparty computation.) The result is another secret-shared dataset, given as a matrix that contains the result of applying the "map" operation on the dataset; or, for our second variant, a secret-shared vector that contains the result of applying a "map-reduce" operation on the dataset.

Our method can be based on any standard technique for performing multi-party computation between the workers. We require the ability to compute on numbers in a given ring with the primitive operations of addition and multiplication. As is standard in the literature, we describe multi-party computation algorithms as normal algorithms, except that secret-shared values are between brackets, e.g., $[\![x]\!]$, and operations like $[\![x]\!] \cdot [\![y]\!]$ induce a cryptographic protocol between the workers implementing the given operation. Examples of such frameworks are passively secure MPC based on Shamir secret sharing, as described in [35]; or the SPDZ family of protocols, as described in [34, 33, 65].

We further assume four higher-level operations that allow to access array elements at sensitive indices:

- $[\![i]\!] \leftarrow \mathsf{Ix}(ix)$ returns a "secret index" $[\![i]\!]$: a representation of array index ix as one or more secret-shared values
- $r \leftarrow \mathsf{IxGet}([\![\mathbf{M}]\!]; [\![i]\!])$ returns the row in the secret-shared matrix $[\![\mathbf{M}]\!]$ pointed to by secret index $[\![i]\!]$
- $[\![\mathbf{M}'']\!] \leftarrow \mathsf{IxSet}([\![\mathbf{M}]\!]; [\![\mathbf{M}']\!]; [\![i]\!])$ returns secret shares of matrix $[\![\mathbf{M}]\!]$ with the row pointed to by secret index $[\![i]\!]$ replaced by the respective row from $[\![\mathbf{M}']\!]$

Protocol 6 Secret indexing of indices $1,\ldots,n$ based on arrays

1: **function** Ix(ix) ▷ return secret index representation of ix
2: **return** $[\![0]\!],\ldots,[\![0]\!],[\![1]\!],[\![0]\!],\ldots,[\![0]\!]$ ▷ one at ixth location
3: **function** IxGet($[\![\mathbf{M}]\!];\Delta$) ▷ return row of $\mathbf{M}\in\mathbb{F}^{n\times k}$ indicated by Δ
4: **return** $(\sum_{u=1}^{n}[\![\Delta_u]\!]\cdot[\![\mathbf{M}_{u,v}]\!])_{v=1,\ldots,k}$
5: **function** IxSet($[\![\mathbf{M}]\!];[\![\mathbf{M'}]\!];[\![ix]\!]$) ▷ return $\mathbf{M}\in\mathbb{F}^{n\times k}$ with Δth row from $\mathbf{M'}$
6: **return** $([\![\mathbf{M}_{u,v}]\!]+[\![\Delta_u]\!]\cdot([\![\mathbf{M'}_{u,v}]\!]-[\![\mathbf{M}_{u,v}]\!]))_{u=1,\ldots,n;v=1,\ldots,k}$
7: **function** IxCondUpdate($[\![\Delta]\!];[\![\delta]\!]$) ▷ return secret index repr. of $\Delta+\delta$, $\delta\in\{0,1\}$
8: **return** $([\![\Delta_i]\!]+[\![\delta]\!]\cdot([\![\Delta_{i-1}]\!]-[\![\Delta_i]\!]))_{i=1,\ldots,n}$

- $[\![i']\!]\leftarrow$ IxCondUpdate($[\![i]\!],[\![\delta]\!]$)) returns a secret index pointing to the same index if $[\![\delta]\!]=0$, and to the next index if $[\![\delta]\!]=1$

Multiple ways of implementing these operations based on an existing MPC framework are known in the literature. We use an adaptation to matrices of the vector indexing techniques from [35], which we have spelled out explicitly in Algorithm 6. An alternative technique is obtained by adapting the secret vector indexing techniques from [37].

3.2 The filtered map procedure

Algorithm 7 provides details of our mapping procedure. The algorithm takes as arguments the function f of the mapping, the simplified function g and predicate ϕ specifying when it can be used, an upper bound N on the number of items for which ϕ does not hold, and a vector z containing some default value on which f can be applied (but whose results are not used).

First, a vector $[\![v]\!]$ is computed that contains a one for each row of $[\![\mathbf{M}]\!]$ where ϕ is *not* satisfied, and a zero where ϕ is satisfied (line 3).

Next, given matrix $[\![\mathbf{M}]\!]$ and vector $[\![v]\!]$ the algorithm builds a matrix $[\![\mathbf{M}']\!]$ with all 1-marked rows of $[\![\mathbf{M}]\!]$ as follows. First, each row of $[\![\mathbf{M}']\!]$ is initialised to $[\![v]\!]$ (line 5). Next, $[\![\mathbf{M}']\!]$ is filled in by going through $[\![\mathbf{M}]\!]$ row-by-row. By the update of secret index $[\![j]\!]$ in line 10, whenever $[\![v_i]\!]=1$, $[\![j]\!]$ points to the row number of $[\![\mathbf{M}']\!]$ where the current row of $[\![\mathbf{M}]\!]$ is supposed to go. Matrix $[\![\Delta\mathbf{M'}]\!]$ is set that is equal to $[\![\mathbf{M'}]\!]$ if $[\![v_i]\!]$ is zero, and consists of N copies of the ith row of $[\![\mathbf{M}]\!]$ if $[\![v_i]\!]$ is one (line 8). The $[\![j]\!]$th row of $[\![\Delta\mathbf{M'}]\!]$ is then copied to matrix $[\![\mathbf{M'}]\!]$ (line 9). Note that if $[\![v_i]\!]=0$ then $[\![\mathbf{M'}]\!]$ does not change; otherwise its $[\![j]\!]$th row is set to the ith row of $[\![\mathbf{M}]\!]$, as was supposed to happen.

Now, function f is applied to all elements of the smaller matrix $[\![\mathbf{M'}]\!]$ (line 12) and function g is applied to all elements of $[\![\mathbf{M}]\!]$ (line 13).

Finally, the results of applying f to $[\![\mathbf{M'}]\!]$ are merged with the results of applying g to $[\![\mathbf{M}]\!]$. The algorithm goes through all rows of $[\![\mathbf{M}]\!]$, where secret index $[\![j]\!]$ keeps track of which row of $[\![\mathbf{N'}]\!]$ should be written to $[\![\mathbf{N}]\!]$ if $[\![v_i]\!]=1$ (line 19). The respective row is retrieved from $[\![\mathbf{N'}]\!]$ (line 17); and the ith row of $[\![\mathbf{N}]\!]$ is overwritten with that row if $[\![v_i]\!]=1$ or kept as-is if $[\![v_i]\!]=0$ (line 18).

3.3 The filtered map-reduce procedure

Algorithm 8 provides the details for the map-reduce procedure. The first steps, to check ϕ and obtain a compressed matrix $[\![\mathbf{M}']\!]$ (lines 2–10), are identical to above. In this case, f is applied to $[\![\mathbf{M}']\!]$ (line 12)

Protocol 7 Filtered map with complex function f, simple function g, predicate ϕ, upper bound N, default value z

1: **function** FilteredMap($f, g, \phi, N, z; [\![\mathbf{M}]\!]$)
2: ▷ compute vector $[\![v]\!]$ containing ones when predicate ϕ is not satisfied
3: **for** $i = 1, \ldots, |[\![\mathbf{M}]\!]|$ **do** $[\![v_i]\!] \leftarrow 1 - \phi([\![\mathbf{M}_i]\!])$
4: ▷ compress dataset $[\![\mathbf{M}]\!]$ to items $[\![\mathbf{M}']\!]$ not satisfying ϕ
5: **for** $i = 1, \ldots, N$ **do** $[\![\mathbf{M}'_i]\!] \leftarrow z$
6: $[\![j]\!] \leftarrow \mathsf{Ix}(0)$
7: **for** $i = 1, \ldots, |[\![\mathbf{M}]\!]|$ **do**
8: $[\![\Delta\mathbf{M'}]\!] \leftarrow ([\![\mathbf{M}'_{u,v}]\!] + [\![v_i]\!] \cdot ([\![\mathbf{M}_{i,v}]\!] - [\![\mathbf{M}'_{u,v}]\!]))_{u=1,\ldots,N;v=1,\ldots,k}$
9: $[\![\mathbf{M}']\!] = \mathsf{IxSet}([\![\mathbf{M}']\!]; [\![\Delta\mathbf{M'}]\!]; [\![j]\!])$
10: $[\![j]\!] \leftarrow \mathsf{IxCondUpdate}([\![j]\!], [v_i])$
11: ▷ apply f to compressed dataset, g to full dataset
12: **for** $i = 1, \ldots, N$ **do** $[\![\mathbf{N}'_i]\!] \leftarrow f([\![\mathbf{M}'_i]\!])$
13: **for** $i = 1, \ldots, |[\![\mathbf{M}]\!]|$ **do** $[\![\mathbf{N}_i]\!] \leftarrow g([\![\mathbf{M}_i]\!])$
14: ▷ decompress results from $[\![\mathbf{N}']\!]$ back into $[\![\mathbf{N}]\!]$
15: $[\![j]\!] \leftarrow \mathsf{Ix}(0)$
16: **for** $i = 1, \ldots, |[\![\mathbf{M}]\!]|$ **do**
17: $[\![c]\!] \leftarrow \mathsf{IxGet}([\![\mathbf{N}']\!]; [\![j]\!])$
18: $[\![\mathbf{N}_i]\!] \leftarrow ([\![\mathbf{N}_{i,j}]\!] + [\![v_i]\!] \cdot ([\![c_j]\!] - [\![\mathbf{N}_{i,j}]\!]))_{j=1,\ldots,k}$
19: $[\![j]\!] \leftarrow \mathsf{IxCondUpdate}([\![j]\!], [v_i])$
20: **return** $[\![\mathbf{N}]\!]$

but there is no need to apply g to $[\![\mathbf{M}]\!]$. Instead, the result is reduced with $\oplus$ and the result returned (line 14).

3.4 Discussion and Possible Extensions

Obtaining upper bounds Our algorithms assume that an upper bound N is available on the number of items in the dataset not satisfying the predicate ϕ. In some situations, such an upper bound may already be available. For example, in our case study below, the "map" computation is combined with the disclosure of an aggregated version of the dataset, from which an upper bound can be determined. In other situations, an upper bound may not be available but revealing it may not be considered a privacy problem. In this case, after determining the vector $[\![v]\!]$, its sum $\sum[\![v_i]\!]$ can be opened up by the workers and used as value for N. (As an alternative, the sum can be rounded or perturbed so as not reveal its exact value.) In yet other situations, a likely upper bound may be available but it may be violated. In such a case, $\sum[\![v_i]\!]$ can be computed and compared to the supposed upper bound, only leaking the result of that comparison disclosed.

Executing g only on mapped items Here, g is executed on *all* items in the dataset, whereas its result is only used in case ϕ is satisfied. If, apart from an upper bound on the number of items not satisfying ϕ, there is also a lower bound (i.e., an upper bound on the number of items satisfying ϕ), then by a natural extension, it is also possible to compute g just on those items at the expense of making compression/decompression more expensive. In cases where g is relatively complex, this can pay off.

Protocol 8 Filtered map-reduce with complex function f, predicate ϕ, operator $\oplus$, upper bound N, default value z

function $\mathsf{FilteredMapReduce}(f, \phi, \oplus, N, z; [\![\mathbf{M}]\!])$
  ▷ for each item in dataset, check whether predicate ϕ is satisfied
  for $i = 1, \ldots, |[\![\mathbf{M}]\!]|$ **do** $[\![v_i]\!] \leftarrow 1 - \phi([\![\mathbf{M}_i]\!])$
  ▷ compress dataset $[\![\mathbf{M}]\!]$ to items $[\![\mathbf{M}']\!]$ not satisfying ϕ
  for $i = 1, \ldots, N$ **do** $[\![\mathbf{M}'_i]\!] \leftarrow z$
  $[\![j]\!] \leftarrow \mathsf{Ix}(0)$
  for $i = 1, \ldots, |[\![\mathbf{M}]\!]|$ **do**
    $[\![\Delta\mathbf{M'}]\!] \leftarrow ([\![\mathbf{M}'_{u,v}]\!] + [\![v_i]\!] \cdot ([\![\mathbf{M}_{i,v}]\!] - [\![\mathbf{M}'_{u,v}]\!]))_{u=1,\ldots,N;v=1,\ldots,k}$
    $[\![\mathbf{M}']\!] = \mathsf{IxSet}([\![\mathbf{M}']\!]; [\![\Delta\mathbf{M'}]\!]; [\![j]\!])$
    $[\![j]\!] \leftarrow \mathsf{IxCondUpdate}([\![j]\!], [v_i \neq 0])$
  ▷ apply f to compressed dataset
  for $i = 1, \ldots, N$ **do** $[\![\mathbf{N}'_i]\!] \leftarrow f([\![\mathbf{M}'_i]\!])$
  ▷ Reduce results using $\oplus$
  return $[\![\mathbf{N}'_1]\!] \oplus \ldots \oplus [\![\mathbf{N}'_N]\!]$

Blockwise application For large datasets, instead of applying our procedures to the whole dataset, it is more efficient to divide the dataset into smaller blocks and apply our map function to these smaller blocks. This is because the indexing functions used in our compression and decompression algorithm typically scale linearly in both the size of the non-compressed and compressed datasets. However, dividing into smaller blocks requires upper bounds for each individual block to be known, as opposed to one overall upper bound. This decreases privacy insofar as these upper bounds are not already known for other reasons. In this sense, our technique allows a trade-off between speed and privacy (where a block size of 1 represents the previously mentioned state-of-the-art alternative to reveal predicate ϕ for each item in the data set).

Flexible application While our technique avoids unnecessary executions of f, it does so at the expense of additional evaluations of g, ϕ, and the compression and decompression procedures. Hence, if the upper bound N is not small enough, using our technique does not save time. (For instance, in the case study below, it only saves time if at most five out of ten items do not satisfy ϕ.) If the execution times of the various components are known, then based on the upper bound a flexible decision can be made whether to perform a traditional or filtered map. If these times are not known beforehand, they can be measured on-the-go. (And, if the upper bound is zero, then compression/decompression is skipped.)

Application to Evolving Datasets Consider a setting in which we want to keep track of the result of a "map" operation on an evolving dataset (that is not necessarily sparse). That is, after running a map operation on $[\![\mathbf{M}]\!]$ returning $[\![\mathbf{N}]\!]$, e.g., $[\![\mathbf{N}]\!] \leftarrow \mathsf{FilteredMap}(\ldots, [\![\mathbf{M}]\!])$, the dataset $[\![\mathbf{M}]\!]$ is modified into a slightly changed dataset $[\![\mathbf{M}']\!]$, and we now wish to obtain the result $[\![\mathbf{N}']\!]$ of applying the map to $[\![\mathbf{M}']\!]$. We do not assume sparsity of $\mathbf{M}$ but we do assume that there is a known (and low) upper bound on the number of changed rows.

This problem occurs, for instance, when performing the "sequential analysis" variant of the Kaplan-Meier survival analysis discussed in the next section. In this case, the survival analysis is performed on a dataset containing a number of patients; if the results of the survival analysis are inconclusive, the the

number of patients is increased and the test is performed again. This test involves a map on a dataset containing event information for each time point; if few patients are added, then few of these rows will have been changed.

This problem is solved with a slightly modified version of FilteredMap. Assume that N is an upper bound on the number changes from $[\![\mathbf{M}]\!]$ to $[\![\mathbf{M}']\!]$. In Algorithm 7, instead of computing $[\![v_i]\!]$ based on ϕ, we set $[\![v_i]\!]$ to one whenever $[\![\mathbf{M_i}]\!] \neq [\![\mathbf{M_i}]\!]$, i.e., $[\![v_i]\!] \leftarrow 1 - \mathsf{eqz}([\![\mathbf{M}_{i,1}]\!] - [\![\mathbf{M'}_{i,1}]\!]) \cdot \ldots \cdot ([\![\mathbf{M}_{i,k}]\!] - [\![\mathbf{M'}_{i,k}]\!])$ using function eqz described in the next section. In line 13, rather than computing $[\![\mathbf{N}_i]\!]$ using function g, we take the previous map result $[\![\mathbf{N}]\!]$.

Also FilteredMapReduce can be made to work in this setting. Suppose that $[\![v]\!]$ is the running total of applying filtered map-reduce on $[\![\mathbf{M}]\!]$ (e.g., as a result of applying FilteredMapReduce), and suppose that $[\![\mathbf{M}']\!]$ differs from $[\![\mathbf{M}]\!]$ at at most N entries. We can get a new running total $[\![v']\!]$ via a similar approach. (Unfortunately this requires $2N$ evaluations of f instead of N. This is because we do not just need to add the new result of apply f to v: we also need to subtract the old result, which we thus need to re-compute.) In detail, we compute $[\![v_i]\!]$ as in the preceding paragraph. We apply lines 4–10 both to $[\![\mathbf{M}]\!]$ to obtain $[\![\mathbf{S}]\!]$, and to $[\![\mathbf{M'}]\!]$ to obtain $[\![\mathbf{S}']\!]$. In line 12, we compute $[\![\mathbf{N}'_i]\!] \leftarrow f([\![\mathbf{S}'_i]\!]) \ominus f([\![\mathbf{S}_i]\!])$. Finally, in line 14 we return $[\![v]\!] \oplus [\![\mathbf{N}'_1]\!] \oplus \ldots \oplus [\![\mathbf{N}'_N]\!]$.

3.5 Use Case: Kaplan-Meier Survival Analysis

We now present a concrete case study in which the above techniques improve the performance of a "map" operation (specifically, the filtered map-reduce). The case study is to perform Kaplan-Meier survival analysis.

Kaplan-Meier Survival Analysis The Kaplan-Meier estimator is an estimation of the survival function (i.e., the probability that a patient survives beyond a specified time) based on lifetime data. The estimated probability p_i at a given time i is given as $p_i = \prod_{j \leq i}(n_j - d_j)/n_j$, where n_j is the number of patients still in the study just before time j and d_j is the number of deaths at time j; the product is over all time points where a death occurred. (n_j decreases not just by deaths but also by people dropping out of the study for other reasons.)

A simple statistical test to decide if two Kaplan-Meier estimates are statistically different is the so-called Mantel-Haenzel logrank test. For instance, this is the test performed by R's `survdiff` call[1]. Given values $n_{j,1}, n_{j,2}, d_{j,1}, d_{j,2}$ at each time point t, define:

$$E_{j,1} = \frac{(d_{j,1} + d_{j,2}) \cdot n_{j,1}}{n_{j,1} + n_{j,2}}; \quad V_j = \frac{n_{j,1} n_{j,2} (d_{j,1} + d_{j,2})(n_{j,1} + n_{j,2} - d_{j,1} - d_{j,2})}{(n_{j,1} + n_{j,2})^2 \cdot (n_{j,1} + n_{j,2} - 1)};$$

$$X = \frac{\sum_j E_{j,1} - \sum_j d_{j,1})}{\sum_j V_j}.$$

The null hypothesis, i.e., the hypothesis that the two curves represent the same underlying survival function, corresponds to $X \approx \chi^2_1$. This null hypothesis is rejected (i.e., the curves are different) if $1 - \mathrm{cdf}(X) > \alpha$, where cdf is the cumulative density function of the χ^2_1 distribution and, e.g., $\alpha = 0.05$.

Note that the computation of this statistical test can be performed using a map-reduce operation. Namely, we map each tuple $(n_{j,1}, n_{j,2}, d_{j,1}, d_{j,2})$ to $(E_{j,1}, V_i, d_{j,1})$ and reduce these values using point-wise summation to obtain $(\sum E_{j,1}, \sum V_j, \sum d_{j,1})$; and we use these values to compute X. Moreover, note

[1] Alternatively, R can perform a variant of this test: the "Cox-Mantel logrank test", which is much more complicated to compute.

that, under the easy to establish criterion $\phi := (d_{j,1}, d_{j,2}) = (0,0)$, we have that $(E_{j,1}, V_i, d_{j,1}) = (0,0,0)$ (the neutral element under point-wise summation), so the conditions under which we can apply filtered map-reduce are satisfied. As default value, $z = (n_{j,1}, n_{j,2}, d_{j,1}, d_{j,2}) = (1,1,0,0)$ can be used.

Anonymized Survival Graph and Upper Bounds In SODA deliverable D4.1, requirements for demonstrators, we propose a work-flow for medical research in which the researcher inspects anonymized (e.g., aggregated) patient information to formulate hypotheses that are then tested in a privacy-friendly way on the exact, non-anonymized data. In the case of Kaplan-Meier, the non-anonymized data are the values n_j and d_j at each time. We propose to anonymize this data by merging different time points. In particular, a block of N consecutive time points $(n_i, d_i)_{i=1,\ldots,N}$ are anonymized to one time point (n,d) with $n = n_1$, $d = \sum d_i$.

Note that this anonymized survival data enables us to establish an upper bound on the number of time points for which the above ϕ does not hold. Namely, give anonymized time points (n,d), (n',d'), the number of points in the block corresponding to (n,d) is at most $n - n'$: the number of people that dropped out during that time interval. Hence, for each block we have an upper bound, enabling us to use block-wise application of our map-reduce algorithm as discussed above.

Performing the Kaplan-Meier Statistical Test We now provide the details of performing the statistical test on Kaplan-Meier survival data. Apart from the basic MPC framework discussed above, we also assume availability of the following:

- A function $[\![c]\!] \leftarrow \mathsf{Div}([\![a]\!], [\![b]\!], L)$ that, given secret-shared $[\![a]\!]$ and $[\![b]\!]$, returns secret-shared $[\![c]\!]$ such that $a/b \approx c \cdot 2^{-L}$, i.e., c is a fixed-point representation of a/b with L bits precision. Such an algorithm can be obtained by straightforward adaptation of division algorithms from [35] or [15]. By convention, we write $[\![a]\!]^1$, $[\![a]\!]^2$ for a secret-share representing a fixed-point value with precision BITS_1, BITS_2 defined by the application.

- An operator $[\![b]\!] \leftarrow [\![a]\!] \gg L$ that shifts secret-shared value a to the right by L bits, as also found in [35] or [15].

- A function $[\![fl]\!] \leftarrow \mathsf{eqz}([\![x]\!])$ that sets $fl = 1$ if $x = 0$, and $fl = 0$ otherwise. A protocol to implement this is described in [35].

Given these primitives, it is straightforward to implement the row-wise operation for the Kaplan-Meier test, i.e., the function f for the map-reduce operation, as shown in Algorithm 9. The algorithm to evaluate ϕ, i.e., the function that computes which rows do not contribute to the test, is shown in Algorithm 10.

The overall algorithm for performing the logrank test is shown in Algorithm 11. First, as discussed above, we compute anonymized survival data (lines 1–5): for each S-sized block we take the number of participants from the first time point (line 4) and the sum of deaths from all time points (line 5). Then, for each block, the upper bounds on the number of events is computed (line 7) and the FilteredMapReduce function is applied to obtain the contributions of those time points to the overall test statistic (line 9). This information is summed together, and from that the test statistic is computed (lines 10–17).

Implementation and Results To test the effectiveness of our approach, we have made a prototype implementation of the above system. The multi-party computation framework has been instantiated by FRESCO (https://github.com/aicis/fresco), using their SPDZ back-end for two parties. This framework provides the MPC functionality needed, as discussed in Section 3.1. Concerning the

Protocol 9 Log test inner loop

Input: $[\![\mathbf{d}_{i,1}]\!], [\![\mathbf{d}_{i,2}]\!], [\![\mathbf{n}_{i,1}]\!], [\![\mathbf{n}_{i,2}]\!]$ survival data at time point i
Output: $([\![e_i]\!]^1, [\![v_i]\!]^1, [\![d_i]\!])$ contributions to $\sum_j E_{j,1}$, $\sum_j V_j$, $\sum_j d_{j,1}$ for test statistic X

1: **function** $f([\![\mathbf{d}_{i,1}]\!], [\![\mathbf{d}_{i,2}]\!], [\![\mathbf{n}_{i,1}]\!], [\![\mathbf{n}_{i,2}]\!])$
2: $\quad [\![ac]\!] \leftarrow [\![\mathbf{d}_{i,1}]\!] + [\![\mathbf{d}_{i,2}]\!]$
3: $\quad [\![bd]\!] \leftarrow [\![\mathbf{n}_{i,1}]\!] + [\![\mathbf{n}_{i,2}]\!]$
4: $\quad [\![frc]\!]^1 \leftarrow \mathsf{Div}([\![ac]\!]; [\![bd]\!]; \mathrm{BITS\_1})$
5: $\quad [\![e_i]\!]^1 \leftarrow [\![frc]\!]^1 \cdot [\![\mathbf{n}_{i,1}]\!]$
6: $\quad [\![vn]\!] \leftarrow [\![\mathbf{n}_{i,1}]\!] \cdot [\![\mathbf{n}_{i,2}]\!] \cdot [\![ac]\!] \cdot ([\![bd]\!] - [\![ac]\!])$
7: $\quad [\![vd]\!] \leftarrow [\![bd]\!] \cdot [\![bd]\!] \cdot ([\![bd]\!] - 1)$
8: $\quad [\![v_i]\!]^1 \leftarrow \mathsf{Div}([\![vn]\!]; [\![vd]\!]; \mathrm{BITS\_1})$
9: $\quad$ **return** $([\![e_i]\!]^1, [\![v_i]\!]^1, [\![d_i]\!])$

Protocol 10 Log test criterion

Input: $[\![\mathbf{d}_{i,1}]\!], [\![\mathbf{d}_{i,2}]\!], [\![\mathbf{n}_{i,1}]\!], [\![\mathbf{n}_{i,2}]\!]$ survival data at time point i
Output: $[\![fl]\!] = 1$ if time point does not provide contribution to test statistic

1: **function** $\phi([\![\mathbf{d}_{i,1}]\!], [\![\mathbf{d}_{i,2}]\!], [\![\mathbf{n}_{i,1}]\!], [\![\mathbf{n}_{i,2}]\!])$
2: $\quad [\![fl]\!] \leftarrow \mathsf{eqz}([\![\mathbf{d}_{i,1}]\!] + [\![\mathbf{d}_{i,2}]\!])$
3: $\quad$ **return** $[\![fl]\!]$

additional MPC functionality required for Kaplan-Meier, as discussed in Section 3.5: we used the division protocol from [15], but adapted it to perform right-shifts after every iteration so that it works for smaller moduli; for right-shifting and zero testing we used the protocols provided by FRESCO. We used constants BITS_1 = 23, BITS_1 = 30.

As a performance metric, we use an estimate of the preprocessing time required for the computation. The SPDZ protocol used, while performing a computation, consumes certain pre-processed data (in particular, so-called multiplication triples and pre-shared random bits) that needs to be generated prior to performing the computation. With state-of-the-art tools, the effort for preprocessing is one or more orders of magnitude more than the effort for the computation itself [65]; therefore, preprocessing effort is a realistic measure of overall effort. To estimate preprocessing time, we keep track of the amount of pre-processed data needed during the computation; we then multiply this with the cost per pre-processed item, which we obtained by simulating both preprocessing parties in one virtual machine on a modern laptop.[2]

Figure 3.1 compares a traditional MPC-based map approach to our approach for the Kaplan-Meier case study. Assuming a list of ten time points for which an upper bound is available (for instance, because anonymized data is released in groups of ten time points), the graph compares normal map (the top row) with our filtered map for various values of the upper bound. As can be seen, applying the filter (i.e., determining which rows do not contribute to the final result) takes virtually no time. Afterwards, time needed for compression, mapping on the compressed list, and decompression, each increase linearly with the upper bound. When the upper bound is six, the overhead of compressing and decompressing becomes so large that it is faster to perform a direct map (if decompression is not needed, compress+map is still cheaper for upper bound six but not for upper bound seven). The overall result on the overall Kaplan-Meier computation on a representative dataset (the "btrial" set supplied

[2] Actual costs may be different in different deployments, but this gives an idea of the relative performance between using the normal map and our optimised map.

Protocol 11 Logrank test on survival curves, using filtered map-reduce

Input: $[\![\mathbf{n}]\!], [\![\mathbf{d}]\!]$ t-by-2 matrices of lifetime data for two populations; S step size
Output: $\mathbf{N}, \mathbf{D}$ anonymized data, p p-value for hypothesis of same survival function

1: **for** $j \leftarrow 1, \ldots, \lceil t/S \rceil + 1$ **do** ▷ generate anonymized data
2: $b \leftarrow (j-1)S+1$; $e \leftarrow jS$
3: ▷ by convention, $[\![\mathbf{n}]\!]$ is extended with copies of its last row and $[\![\mathbf{d}]\!]$ by zeros
4: $\mathbf{N}_{j,1} = \mathsf{Open}([\![\mathbf{n}_{b,1}]\!])$ $\mathbf{N}_{j,2} = \mathsf{Open}([\![\mathbf{n}_{b,2}]\!])$
5: $\mathbf{D}_{j,1} = \mathsf{Open}(\sum_{i=b}^{e} [\![\mathbf{d}_{i,1}]\!])$; $\mathbf{D}_{j,2} = \mathsf{Open}(\sum_{i=b}^{e} [\![\mathbf{d}_{i,2}]\!])$
6: **for** $j \leftarrow 1, \ldots, \lceil t/S \rceil$ **do** ▷ compute contributions for each block
7: $b \leftarrow (j-1)S+1$; $e \leftarrow jS$; $N \leftarrow N_{j,1} + N_{j,2} - N_{j+1,1} - N_{j+1,2}$
8: $[\![\mathbf{M}]\!] \leftarrow \{[\![\mathbf{d}_{i,1}]\!], [\![\mathbf{d}_{i,2}]\!], [\![\mathbf{n}_{i,1}]\!], [\![\mathbf{n}_{i,2}]\!]\}_{i=b,\ldots,e}$
9: $([\![e_j]\!]^1, [\![v_j]\!]^1, [\![d'_j]\!]) \leftarrow \mathsf{FilteredMapReduce}(f, \phi, +, N, (0,0,1,1); [\![\mathbf{M}]\!])$
10: $[\![dtot]\!] \leftarrow \sum_{j=1}^{\lceil t/S \rceil} [\![d'_j]\!]$; $[\![dtot]\!]^1 \leftarrow [\![dtot]\!] \ll \text{BITS\_1}$
11: $[\![etot]\!]^1 \leftarrow \sum_{j=1}^{\lceil t/S \rceil} [\![e_j]\!]^1$; $[\![vtot]\!]^1 \leftarrow \sum_{j=1}^{\lceil t/S \rceil} [\![v_j]\!]^1$
12: $[\![dmi]\!]^1 \leftarrow [\![dtot]\!]^1 - [\![vtot]\!]^1$
13: $[\![chi0]\!]^2 \leftarrow \mathsf{Div}([\![dmi]\!]^1; [\![vtot]\!]^1; \text{BITS\_2})$
14: $[\![chi]\!]^{12} \leftarrow [\![chi0]\!]^2 \cdot [\![dmi]\!]^1$
15: $[\![chi]\!]^1 \leftarrow [\![chi]\!]^{12} \gg \text{BITS\_2}$
16: $chi^1 \leftarrow \mathsf{Open}([\![chi]\!]^1)$
17: $p \leftarrow 1 - \mathsf{cdf}\chi_1^2(chi^1 \cdot 2^{-\text{BITS\_1}})$
18: **return** $\mathbf{N}, \mathbf{D}, p$

with the R statistics package) is a time decrease of 51% on the map-reduce operation, which is the bulk of the overall computation.

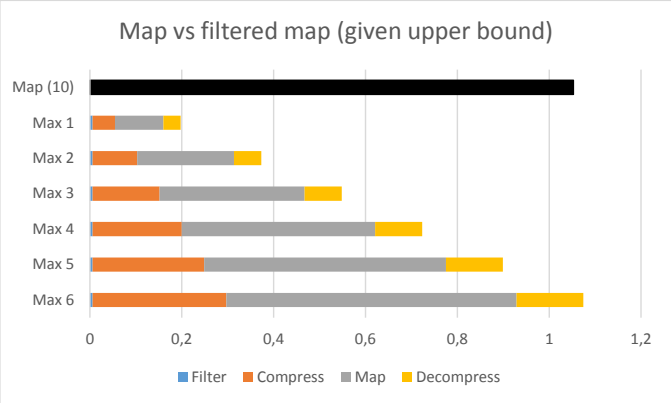


Figure 3.1: Performance of normal vs filtered map on length-10 Kaplan-Meier survival database, in terms of estimated preprocessing time

Chapter 4

Combining Private Set-Intersection with Secure Two-Party Computation

Private Set-Intersection (PSI) is one of the most popular and practically relevant secure two-party computation (2PC) tasks. Therefore, designing special-purpose PSI protocols (which are more efficient than generic 2PC solutions) is a very active line of research. In particular, a recent line of work has proposed PSI protocols based on oblivious transfer (OT) which, thanks to recent advances in OT-extension techniques, is nowadays a very cheap cryptographic building block.

Unfortunately, these protocols cannot be plugged into larger 2PC applications since in these protocols one party (by design) learns the output of the intersection. Therefore, it is not possible to perform secure *post-processing* of the output of the PSI protocol.

In this chapter we describe a novel and efficient OT-based PSI protocol that produces an "encrypted" output that can therefore be later used as an input to other 2PC protocols. In particular, the protocol can be used in combination with all common approaches to 2PC including garbled circuits, secret sharing and homomorphic encryption. Thus, the protocol can be combined with the right 2PC techniques to achieve more efficient protocols for computations of the form $z = f(X \cap Y)$ for arbitrary functions f.

The protocol and the content of this chapter first appeared as a paper with title "Combining Private Set-Intersection with Secure Two-Party Computation" by Michele Ciampi and Claudio Orlandi. The paper was published and presented at SCN 2018 [22].

4.1 Introduction

Private Set-Intersection (PSI) is one of the most practically relevant *secure two-party computation* (2PC) tasks. In PSI two parties hold two sets of strings X and Y, respectively. At the end of the protocol one (or both) party should learn the intersection of the two sets $Z = X \cap Y$ and nothing else about the input of the other party.

There are many real-world applications in which PSI is required. As an example, when mobile users install messaging apps, they need to discover whom among their contacts (from their address book) is also using the app, in order to be able to start communicating seamlessly with them. Doing so requires users to learn the intersection of their contact list with the list of registered users of the service which is stored at the server side. This is typically done by having users send their contact list to the server that can then compute the intersection and return the result to the user. Unfortunately this solution is very problematic not only for the privacy of the user, but for the privacy of the users' contacts as well! In particular, some of the people in the contact list might not want their

phone number being transferred and potentially stored by the server, but they have no control over this.[1] Note that this is not just a theoretically interesting problem and that Signal (one of the most popular end-to-end encrypted messaging app) has recently recognized this as being a real problem and offered partial solutions to it.[2] PSI has many other applications, including computing intersections of suspect lists, private matchmaking (comparing interests), *testing human genome* [5], *privacy-preserving ride-sharing* [49, 86], *botnet detection* [81], *advertisement conversion rate* [54] and many more.

From a feasibility point of view, PSI is just a special case of 2PC and therefore any generic 2PC protocol (such as [103, 47]) could be used to securely evaluate PSI instances as well. However, since PSI is a natural functionality that can be applied in numerous real-world applications, many efficient protocols for this specific functionality have been proposed, with early results dating back to the 80s [98, 74]. The problem was formally defined in [44] and follow up work increased the efficiency of PSI protocols to have complexity only *linear* in the inputs of the parties [58, 27]. A very recent work shows how to obtain a protocol where communication complexity is linear in the size of the smaller set and logarithmic in the larger set [19].

However, these protocols still require performing expensive public-key operations (e.g., exponentiations) for every element in the input sets. As public-key operations are orders of magnitudes more expensive than symmetric key operations, these protocols are not practically efficient for large input sets. In the meanwhile, generic techniques for 2PC had improved by several orders of magnitude and the question of whether special purpose protocols or generic protocols were most efficient has been debated in [53, 28]. Due to its practical relevance, PSI protocols in the *server-aided* model have been proposed as well [62]. Independent and concurrent works [89, 41] consider the problem of using a PSI protocol to construct more complex functionality in an efficient way. More specifically, [89] provides a way to securely compute many variants of the set intersection functionality using a clever combination of Cuckoo hashing and garbled circuit. The work of Falk et al. [41] focuses on obtaining a PSI protocol that is efficient in terms input communication. In addition, the authors of [41] propose a PSI protocol where the output can be secret shared that has communication complexity of $O(m\lambda \log\log m)$, where λ is the bit-length of the elements and m is the set-size.

The techniques used in this chapter significantly differ from the techniques used in [89, 41]. Our solution avoids the use of garbled circuits and rely on the security and the efficiency of OT and symmetric key encryption schemes.

4.1.1 OT-based PSI

The most efficient PSI protocols today are those following the approach of PSZ [88, 90]. These protocols make extensive use of a cryptographic primitive known as *oblivious transfer (OT)*. While OT provably requires expensive public-key operation, OT can be "extended" as shown by [55, 3, 67] i.e., the few necessary expensive public-key operations can be amortized over a very large number of OT instances, and the marginal cost of OT is only a few (faster) symmetric key operations instead. In particular, improvements in OT-extension techniques directly imply improvements to PSI protocols as shown by e.g., [68, 85].

In a nutshell, the PSZ protocol introduced two important novel ideas to the state of the art of PSI. First, they give an efficient instantiation of the *private set membership protocol* (PSM) introduced

[1] Some apps do not transfer the contact list in cleartext, but a hashed version instead. However, since the domain space of phone numbers is small enough to allow for brute forcing of the hashes, this does not guarantee any real privacy guarantee.

[2] Unfortunately, the Signal team has concluded that current PSI protocols are too inefficient for their application scenario and relied on trusted-hardware instead, in the style of [100]. See https://signal.org/blog/private-contact-discovery/ for more details on this.

in [43] based on OT. Second, they show how to efficiently implement PSI from PSM using hashing techniques. (An overview of their techniques is given below).

4.1.2 Our contribution

The main contribution of this chapter is to give an efficient instantiation of PSM that provides output in encrypted format and can therefore be combined with further 2PC protocols. Our PSM protocol can be naturally combined with the hashing approach of PSZ to give a PSI protocol with encrypted output achieving the same favourable complexity in the input sizes. This enables the combination the efficiency of modern PSI techniques with the potentials of general 2PC. Combining our protocols with the right 2PC post-processing allows more efficient evaluation of functionalities of the form $Z = f(X \cap Y)$ for any function f. Like in PSZ we only focus on semi-honest security. Using the protocol together with an actively secure OT-extension protocol such as [4, 64] would result in a protocol with *privacy* but not *correctness* (i.e., the view of the protocol *without* the output can be efficiently simulated), which is a meaningful notion of security in some settings. PSI protocols with security against malicious adversaries have been proposed in e.g., [51, 94, 95]. It is an interesting open problem to design efficient protocols which are both secure against active (or covert) adversaries and that produce encrypted output. Also, like in PSZ, we only focus on the two-party setting. The recent result of [52] has shown that multiparty set-intersection can be computed efficiently. Extending our result to the multiparty case is an interesting future research direction.

We also compare the computation complexity of our scheme for PSM with all the circuit-based PSI approaches (which can be combined with further postprocessing) proposed in [91]. More precisely, in Table 4.1 we compare our protocol with the protocols of [91] in terms of number of symmetric key operations, and bits exchanged between the parties. The result of this comparison is that our protocol has better performance, in terms of computational complexity, than all the circuit-based PSI approaches considered for our comparison[3]. We refer the reader to the original paper for more details about this comparison.

4.1.3 Improving the efficiency of smart contract protocols

Most of the cryptocurrency systems are built on top of blockchain technologies where miners run distributed consensus whose security is ensured as long as the adversary controls only a minority of the miners. Some cryptocurrency systems allow to run complex programs and decentralized applications on the blockchain. In Ethereum[4] those programs are called smart contracts. Roughly speaking, the aim of a smart contract is to run a protocol and start a transaction to pay a user of the cryptocurrency systems according to the output of the protocol execution. Unfortunately, this interesting feature of the smart contracts does not come for free. Indeed, in order to execute a smart contract, it is required to pay a *gas fee* that depends on the number of instructions of the protocol to be executed. So, higher is the complexity of protocol, higher is the price to pay. In this context a cryptographic protocol that outputs intermediate values in a secret shared way is particularly useful. Suppose that two parties want to securely compute $f(X \cap Y)$ for arbitrary functions f, and reward another party depending on the output of this computation. Instead of writing on a smart contract the entire protocol to compute $f(X \cap Y)$,

[3] The complexity of the protocols proposed in [91] depends upon parameters that are also related to the used hash function. In order to make our comparison fair, we have set these parameters as showed in the first column in Table 10 of [91]. More precisely, the authors of [91] show in that table which parameters are adopted for their empirical efficiency comparison for the case where one set is much greater than the other set (which is exactly the case of PSM).

[4] http://www.ethereum.org.

the two parties could run a sub-protocol Π to obtain a secret share of $\chi = X \cap Y$ without using a smart contract, and then run another sub-protocol Π' to compute $f(\chi)$, this time using a smart contract to enforce the reward policy. Following this approach it is possible to move part of the computation off-chain, thus increasing the performance and, at the same time, decreasing the costs required to execute the smart contract. Moreover, we observe that χ can be reused to compute different functions f'. The scenario described above is particularly interesting if one of the party can be fully malicious, but in this work we will focus on semi-honest security leaving the above as an open question.

	# of sym. key operations	Communication [bits]
Yao SCS [53]	$12\lambda M \log M + 3\lambda M$	$2\lambda Ms(1+3\log M)$
GMW SCS [53]	$12\lambda M \log M$	$6\lambda M(s+2)\log M$
Yao PWC [91]	$4\lambda M + 6393\lambda$	$\lambda(M3s + 3198s + 15,6)$
GMW PWC [91]	$6\lambda M + 9594\lambda$	$\lambda(M4 + 6396 + 2sM + 6396s)$
This work	$4\lambda M + 3\lambda$	$2\lambda Ms + Ms$

Table 4.1: Computation and communication complexity comparison for the PSM case. M represents the size of the set, s is the security parameter and λ is the bit-length of each element.

4.2 Technical overview

4.2.1 Why PSZ and 2PC do not mix

We start with a quick overview of the PSM protocol in PSZ [88, 90], to explain why their protocol inherently reveals the intersection to one of the parties. From a high-level point of view, the protocol is conceptually similar to the PSM protocol from oblivious pseudorandom function (OPRF) of [43], except that the OPRF is replaced with a similar functionality efficiently implemented using OT. For simplicity, here we will use the OPRF abstraction.

The goal of a PSM protocol is the following: the receiver R has input x, and the sender S has input a set Y; at the end of the protocol the receiver learns whether $x \in Y$ or not while the sender learns nothing. The protocol starts by using the OPRF subprotocol, so that R learns $x^* = F_k(x)$ (where k is known to S), whereas S learns nothing about x. Now S evaluates the PRF on her own set and sends the set $Y^* = \{y^* = F_k(y) | y \in Y\}$ to R, who checks if $x^* \in Y^*$ and concludes that $x \in Y$ if this is the case. In other words, we map all inputs into pseudorandom strings and then let one of the parties test for membership "in the clear". Since the party performing the test doesn't have access to the mapping (e.g., the PRF key), this party can only check for the membership of x and no other points (i.e., all elements in $Y^* \setminus \{x^*\}$ are indistinguishable from random in R's view).

From the above description, it should be clear that the PSZ PSM inherently reveals the output to one of the parties. Turning this into a protocol which provides encrypted output is a challenging task. Here is an attempt at a "strawman" solution: we change the protocol such that R still learns the pseudorandom string $x^* = F_k(x)$ corresponding to x, but now S sends a value *for every element in the universe*. Namely, for each i (in the domain of Y) S sends an encryption of whether $i \in Y$ "masked" using $F_k(i)$ e.g., S sends $c_i = F_k(i) \oplus E(i \in Y)$[5]. Now R can compute $c_x \oplus x^* = E(x \in Y)$ i.e., an encrypted version of whether $x \in Y$, which can be then used as input to the next protocol.

[5]The exact format of the "encryption" $E(\cdot)$ would depend on the subsequent 2PC protocol and is irrelevant for this high-level description.

While this protocol produces the correct result, its complexity is exponential in the bit-length of $|x|$, which is clearly not acceptable.

Intuitively, we know that only a polynomial number of c_i's will contain encryptions of 1, and therefore we need to find a way to "compress" all the c_i corresponding to $i \notin Y$ into a single one, to bring the complexity of the protocol back to $O(|Y|)$. In the following, after defining some useful notation, we give an intuitive explanation on how to do that.

4.2.2 Our protocol

We introduce some useful (and informal) notation in order to make easier to understand the ideas behind our construction. We let $Y = \{y_1, \ldots, y_M\}$ be the input set of the sender S, and we assume w.l.o.g., that $|Y| = M = 2^m$.[6] All strings have the same length e.g., $|x| = |y_i| = \lambda$.[7] We will use a special symbol $\perp$ such that $x \neq \perp \; \forall x$. We use a function $\mathsf{Prefix}(x,i)$ that outputs the i most significant bits of x ($\mathsf{Prefix}(x,i) \neq \mathsf{Prefix}(x,j)$ when $i \neq j$ independently of the value of x) and for simplicity we define $\mathsf{Prefix}(Y,i)$ to be the set constructed by taking the i most significant bits of every element in Y.

The protocol uses a symmetric key encryption scheme $\mathsf{Sym} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the additional property that given a key $k \leftarrow \mathsf{Gen}(1^s)$ it is possible to efficiently verify if a given ciphertext is in the range of k (see the original paper for a formal definition).

Finally, the output of the protocol will be one of two strings γ_0, γ_1 chosen by S, respectively denoting $x \notin Y$ and $x \in Y$. The exact format of the two strings depends on the protocol used for post-processing. For instance if the post-processing protocol is based on: 1) *garbled circuits*, then γ_0, γ_1 will be the labels corresponding to some input wire; 2) *homomorphic encryption*, then $\gamma_b = \mathsf{Enc}(pk, b)$ for some homomorphic encryption scheme Enc; 3) *secret-sharing*, then $\gamma_b = s_2 \oplus b$ where s_2 is a uniformly random share chosen by S, so that if R defines its own share as $s_1 = \gamma_b$ then it holds that $s_1 \oplus s_2 = b$.[8]

In order to "compress" the elements of Y we start by considering an undirected graph with a level structure of $\lambda + 1$ levels. The vertices in the last level of this graph will correspond to the elements of Y. More precisely, we associate the secret key $k_{b_\lambda b_{\lambda-1} \ldots b_1}$ of a symmetric key encryption scheme Sym to each element $y = b_\lambda b_{\lambda-1} \ldots b_1 \in Y$. The main idea is to allow the receiver to obliviously navigate this graph in order to get the key $k_{b_\lambda b_{\lambda-1} \ldots b_1}$ if $x = y$, for some $y = b_\lambda b_{\lambda-1} \ldots b_1 \in Y$, or a special key $k^\star$ otherwise. Moreover we allow the receiver to navigate the graph efficiently, that is, every level of the graph is visited only once.

Once a key k is obtained by the receiver, the sender sends $O(|Y|)$ ciphertexts in a such a way that the key obtained by the receiver can decrypt only one ciphertext. Moreover the plaintext of this ciphertext will correspond to γ_0 or γ_1 depending on whether $x \in Y$ or not.

First step: construct the graph G

Each graph level $i \in \{0, \ldots, \lambda\}$ has size at most $|\mathsf{Prefix}(Y,i)| + 1$. More precisely, for every $t = b_\lambda b_{\lambda-1} \ldots b_{\lambda-i} \in \mathsf{Prefix}(Y,i)$ there is a node in the level i of G that contains a key $k_{b_\lambda b_{\lambda-1} \ldots b_{\lambda-i}}$. In addition, in the level i there is a special node, called *sink node* that contains a key $k_i^\star$ (which we refer to as *sink key*). The aim of $k_i^\star$ is to represent all the values that do not belong to $\mathsf{Prefix}(i,Y)$.

[6] Sets can always be padded with dummy elements, but the complexity of the protocol can match M that in practice can be $M \approx 2^{m-1}$.

[7] We can assume λ to be smaller than the (statistical) security parameter s and we will denote the bit decomposition of x by $x = x_\lambda \ldots x_1$.. Otherwise before running the protocol the parties can hash their input down and run the protocol with inputs $h(x)$ and $h(Y) = \{h(y_1), \ldots, h(y_M)\}$. Clearly if $x = y_i$ then $h(x) = h(y_i)$, and for correctness we need that $Pr[h(x) \in h(Y) \wedge x \notin Y] < 2^{-s}$.

[8] Here we use $\oplus$-secret sharing without loss of generality. Any 2-out-2 secret sharing would work here.

Let us now describe how the graph G is constructed. First, for $i = 1, \ldots, \lambda$ the key (for a symmetric key encryption scheme) $k_i^\star$ is generated using the generation algorithm $\mathsf{Gen}(\cdot)$. As discussed earlier the aim of these keys is to represent the elements that do not belong to Y. More precisely, the sink key $k_i^\star$, with $i \in \{1, \ldots, \lambda\}$ represents all the values that do not belong to $\mathsf{Prefix}(Y, i)$ and the key $k_\lambda^\star$ (the last sink key) will be used to encrypt the output γ_0 corresponding to non-membership in the last step of our protocol. Note that if $\mathsf{Prefix}(x, i) \notin \mathsf{Prefix}(Y, i)$ then $\mathsf{Prefix}(x, j) \notin \mathsf{Prefix}(Y, j)$ for all $j > i$. Therefore, once entered in a sink node, the *sink path* is never abandoned and thus the final sink key $k_\lambda^\star$, will be retrieved (which allows recovery of γ_0). Let us now give a more formal idea of how G is constructed.

- The root of G is empty, and in the second level there are two vertices k_0 and k_1 where[9], for $b = 0, 1$

$$k_b = \begin{cases} k \leftarrow \mathsf{Gen}(1^s), & \text{if } b \in \mathsf{Prefix}(Y, 1) \\ k_1^\star, & \text{otherwise} \end{cases}$$

- For each vertex k_t in the level $i \in \{1, \ldots, \lambda\}$ and for $b = 0, 1$ create the node $k_{t||b}$ as follows (if it does not exists) and connect k_t to it.

$$k_{t||b} = \begin{cases} k \leftarrow \mathsf{Gen}(1^s), & \text{if } t||b \in \mathsf{Prefix}(Y, i+1) \\ k_{i+1}^\star, & \text{if } t||b \notin \mathsf{Prefix}(Y, i+1) \\ k_{i+1}^\star, & \text{if } k_t = k_i^\star \end{cases}$$

  We observe that a new node $k_{t||b}$ is generated only when $t||b \in \mathsf{Prefix}(Y, i)$. In the other cases the sink node $k_{i+1}^\star$ is used.

In Fig. 4.1 we show an example of what the graph G looks like for the set $Y = \{010, 011, 110\}$. In this example it is possible to see how, in the 2nd level, all the elements that do not belong to $\mathsf{Prefix}(Y, 2)$ are represented by the sink node $k_2^\star$. Using this technique we have that in the last level of G one node ($k_3^\star$ in this example) is sufficient to represent all the elements that do not belong to Y. Therefore, we have that the last level of G contains at most $|Y| + 1$ elements. We also observe that every level of G cannot contain more than $|Y| + 1$ nodes.

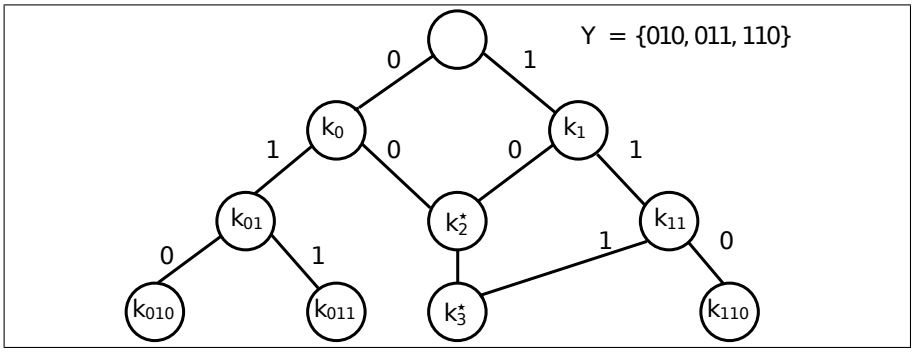


Figure 4.1: Example of how the graph G appears when the sender holds the set Y.

[9] In abuse of notation we refer to a vertex using the key represented by the vertex itself.

Second step: oblivious navigation of G

Let $x = x_\lambda x_{\lambda-1} \dots x_1$ be the receiver's (R's) private input and Y be the sender's (S's) private input. After S constructs the graph G we need a way to allow R to obtain $k_{x_\lambda x_{\lambda-1} \dots x_1}$ if $x \in Y$ and the sink key $k^\star_\lambda$ otherwise. All the computation has to be done in such a way that no other information about the set Y is leaked to the receiver, and as well that no information about x is leaked to the sender. In order to do so we use λ executions of 1-out-of-2 OT. The main idea is to allow the receiver to select which branch to explore in G depending on the bits of x. More precisely, in the first execution of OT, R will receive the key k_{x_λ} iff there exists an element in Y with the most significant bit equal to x_λ, the sink key $k^\star_1$ otherwise. In the second execution of OT, R uses $x_{\lambda-1}$ as input and S uses (c_0, c_1) where c_0 is computed as follows:

- For each key in the second level of G that has the form $k_{t||0}$, the key $k_{t||0}$ is encrypted using the key k_t.
- For every node v in the first level that is connected to a sink node $k^\star_2$ in the second level, compute an encryption of $k^\star_2$ using the key contained in v.
- Pad the input with random ciphertexts up to the upper bound for the size of this layer (more details about this step are provided later).
- Randomly permute these ciphertexts.

The procedure to compute the input c_1 is essentially the same (the only difference is that in this case we consider every key with form $k_{t||1}$ and encrypt it using k_t).

Roughly speaking, in this step every key contained in a vertex u of the second level is encrypted using the keys contained in the vertex v of the previous level that is connected to u. For example, following the graph provided in Fig.4.1, c_0 would be equal to $\{\mathsf{Enc}(k_0, k^\star_2), \mathsf{Enc}(k_1, k^\star_2)\}$ and c_1 to $\{\mathsf{Enc}(k_0, k_{01}), \mathsf{Enc}(k_1, k_{11})\}$.

Thus, after the second execution of OT R receives $c_{x_{\lambda-1}}$ that contains the ciphertexts described above where only one of these can be decrypted using the key k obtained in the first execution of OT. The obtained plaintext corresponds to the key $k_{x_\lambda x_{\lambda-1}}$ if $\mathsf{Prefix}(x,2) \in \mathsf{Prefix}(Y,2)$, to the sink key $k^\star_2$ otherwise. The same process is iterated for all the levels of G. More generally, if $\mathsf{Prefix}(x,j) \in \mathsf{Prefix}(Y,j)$ then after the $j-$th execution of OT R can compute the key $k_{x_\lambda x_{\lambda-1} \dots x_{\lambda-j}}$ using the key obtained in the previous phase. Conversely if $\mathsf{Prefix}(x,j) \notin \mathsf{Prefix}(Y,j)$ then the sink key $k^\star_j$ is obtained by R. We observe that after every execution of OT R does not know which ciphertext can be decrypted using the key obtained in the previous phase, therefore he will try to decrypt all the ciphertext until the decryption procedure is successful. To avoid adding yet more indexes to the (already heavy) notation of our protocol we deal with this using a private-key encryption scheme with efficiently verifiable range. We note that this is not necessary and that when implementing the protocol one can instead use the *point-and-permute technique* [9]. This, and other optimisations and extensions of our protocol, are described in Section 4.3.

Third step: obtain the correct share

In this step S encrypts the output string γ_0 using the key $k^\star_\lambda$ and uses all the other keys in the last level of G to encrypt the output string γ_1.[10] At this point the receiver can only decrypt either the ciphertext

[10] The key $k^\star_\lambda$ could not exists; e.g. if Y contains all the strings of λ bits.

that contains γ_0 if $x \notin Y$ or one (and only one) of the ciphertexts that contain γ_1 if $x \in Y$. In the protocol that we have described so far R does not know which ciphertext can be decrypted using the key that he has obtained. Also in this case we can use a point-and-permute technique to allow R to identify the only ciphertext that can be decrypted using his key.

On the need for padding As describe earlier, we might need to add some padding to the OT sender's inputs. To see why we need this we make the following observation. We recall that in the i-th OT execution the sender computes an encryption of the keys in the level i of the artificial graph G using the keys of the previous level $(i-1)$.[11] As a result of this computation the sender obtains the pair (c_0^i, c_1^i), that will be used as input of the i-th OT execution, where c_0^i (as well as c_1^i) contains a number of encryptions that depends upon the number of vertices on level $(i-1)$ of G. We observe that this leaks information about the structure of G to the receiver, and therefore leaks information about the elements that belong to Y. Considering the example in Fig. 4.1, if we allow the receiver to learn that the 2nd level only contains 3 nodes, then the receiver would learn that all the elements of Y have the two most significant bits equal to either t or t' for some $t, t' \in \{0,1\}^2$ (in Fig.4.1 for example we have $t = 01$ and $t' = 11$; note however that the receiver would not learn the actual values of t and t').

We finally note that the technique described in this section can be seen as a special (and simpler) example of private information retrieval (PIR) by keywords and securely evaluating a branching program. The work of Chor et al. [21] uses a data structure similar to the one described here which is obliviously navigated using a PIR scheme. Secure evaluation of branching programs has previously been considered in [56, 76]: unfortunately these protocols cannot be instantiated using OT-extension and therefore will not lead to practically efficient protocols (the security of these protocols is based on *strong* OT which, in a nutshell, requires the extra property that when executing several OTs in parallel, the receiver should not be able to correlate the answers with the queries beyond correlations which follow from the output).

Full Protocol and Proof of Security. We refer the reader to the original paper for a formal description of the full protocol and its proof of security.

4.3 Optimisations and extensions

4.3.1 Point and permute

In our protocol the receiver must decrypt every ciphertext at every layer to identify the correct one. This is suboptimal both because of the number of decryptions and because encryptions that have efficiently verifiable range necessarily have longer ciphertexts. This overhead can be removed using the standard *point-and-permute technique* [9] which was introduced in the context of garbled circuits. Using this technique we can add to each key in each layer a *pointer* to the ciphertext in the next layer which can be decrypted using this key. This has no impact on security.

4.3.2 One-time pad

It is possible to reduce the communication complexity of our protocol by using *one-time pad encryption* in the last $\log s$ layers of the graph, in the setting where the output values γ^0, γ^1 are such that $|\gamma^b| < s$. For instance, if the output values are bits (in case we combine our PSM with a GMW-style protocol),

[11] The only exception is the first OT execution where just two keys are used as input.

then the keys (and therefore the ciphertexts) used in the last layer of the graph only need to be 1 bit long. Unfortunately, since the keys in the second to last layer are used to mask up to two keys in the last layer, the keys in the second to last layer must be of length 2 and so on, which is why this optimisation only gives benefits in the last $\log s$ layer of the graph.

4.3.3 PSM with secret shared input

Our PSM protocol produces an output which can be post-processed using other 2PC protocols. It is natural to ask whether it is possible to design efficient PSM protocols that also work on encrypted or secret-shared inputs. We note here that our protocol can also be used in the setting in which the input string x is *bit-wise secret-shared* between the sender and the receiver i.e., the receiver knows a share r and the sender knows a share s s.t., $r \oplus s = x$. The protocol does not change for the receiver, who now inputs the bits of $r = r_\lambda, \ldots, r_1$ to the λ one-out-of-two OTs (instead of the bits of x as in the original protocol). The sender, at each layer i, will follow the protocol as described above if $s_i = 0$ and instead swap the inputs to the OT if $s_i = 1$. It can be easily verified that the protocol still produces the correct result and does not leak any extra information.

4.3.4 Keyword search

Our PSM protocol outputs an encryption of a bit indicating whether $x \in Y$ or not. The protocol can be easily modified to output a value dependent on x itself and therefore implement "encrypted keyword search". That is, instead of having only two output strings γ^1, γ^0 representing membership and non-membership respectively, we can have $|Y| + 1$ different output strings (one for each element $y \in Y$ and one for non-membership). This can be used for instance in the context where Y is a database containing id's y and corresponding values $v(y)$, and the output of the protocol should be an encryption of the value $v(x)$ if $x \in Y$ or a standard value $v(\perp)$ if $x \notin Y$. The modification is straightforward: instead of using all the keys in the last layer of the graph to encrypt the same value γ^1, use each key k_y to encrypt the corresponding value $v(y)$ and the sink key (which is used to encrypt γ^0 in our protocol) to encrypt the value $v(\perp)$.

4.3.5 PSI from PSM

We can follow the same approach of Phasing [88, 90] to turn our PSM protocol into a protocol for PSI. Given a receiver with input X and a sender with input Y the trivial way to construct PSI from PSM is to run $|X|$ copies of PSM, where in each execution the receiver inputs a different x from X and where the sender always inputs her entire set Y. As described above, the complexity of our protocol (as the complexity of Phasing) is proportional in the size of $|Y|$, so this naïve approach leads to quadratic complexity $O(|X| \cdot |Y|)$. Phasing deals with this using *hashing* i.e., by letting the sender and the receiver locally preprocess their inputs X, Y before engaging in the PSM protocols. The different hashing techniques are explained and analysed in [91, Section 3]. We present the intuitive idea and refer to their paper for details: in Phasing the receiver uses *Cuckoo hashing* to map X into a vector X' of size $\ell = O(|X|)$ such that all elements of X are present in X' and such that every $x'_i \in X'$ is either an element of X or a special $\perp$ symbol. The sender instead maps her set Y into $\ell = |X'|$ small buckets $Y'_1, \ldots, Y'_\ell$ such that every element $y \in Y$ is mapped into the "right bucket" i.e., the hashing has the property that if $y = x'_i$ for some i then y will end up in bucket Y'_i (and potentially in a few other buckets). Now Phasing uses the underlying PSM protocol to check whether x'_i is a member of Y'_i (for all i's), thus producing the desired result. The overall protocol complexity is now $O(\sum_{i=1}^{l} |X'| \cdot |Y'_i|)$ which (by careful choice

of the hashing parameters) can be made linear in input size i.e., the overall protocol has complexity $O(|X|+|Y|)$.[12] Since this technique is agnostic of the underlying PSM protocol, we can apply the same technique to our PSM protocol to achieve a PSI protocol that produces encrypted output.

4.4 Applications

The major advantage provided by $\Pi^{\in}$ is that the output of the receiver can be an arbitrary value chosen by the sender as a function of x for each value $x \in Y \cup \{\perp\}$. This is in contrast with most of the approaches for set membership, where the value obtained by the receiver is a fixed value (e.g. 0) when $x \in Y$, or some random value otherwise. We now provide two examples of how our protocol can be used to implement more complex secure set operations. The examples show some guiding principles that can be used to design other applications based on our protocol.

Without loss of generality in the following applications only the receiver will learn the output of the computation. Moreover we assume that the size of X and Y is equal to the same value M.[13] Also for simplicity we will describe our application using the naïve PSI from PSM construction with quadratic complexity, but using the PSZ approach, as described in Sec. 4.3, it is possible to achieve linear complexity using hashing techniques. Finally, in both our applications we exploit the fact that additions can be performed locally (and for free) using secret-sharing based 2PC. In applications in which round complexity is critical, the protocols can be redesigned using garbled circuits computing the same functionality, since the garbled circuit can be sent from the sender to the other messages of the protocol. However in this case additions have to be performed inside the garbled circuit.

4.4.1 Computing statistics of the private intersection

Here we want to construct a protocol where sender and receiver have as input two sets, X and Y respectively, and want to compute some statistics on the intersections of their sets. For instance the receiver has a list of id's X and that the sender has a list of of id's Y and some corresponding values $v(Y)$ (thus we use the variant of our protocol for *keyword search* described in Section 4.3). At the end of the protocol the receiver should learn the average of $v(X \cap Y)$ (and not $|X \cap Y|$).

The main idea is the following: the sender and the receiver run M executions of our protocol where the receiver inputs a different x_i from X in each execution. The sender always inputs the same set Y, and chooses the $|Y|+1$ outputs γ_i^y for all $y \in Y \cup \{\perp\}$ for all $i = 1, \ldots, M$ in the following way: γ_i^y is going to contain two parts, namely an arithmetic secret sharing of the bit indicating whether $x_i \in Y$ and an arithmetic secret sharing of the value $v(y)$. The arithmetic secret sharing will be performed using a modulo N large enough such that $N > M$ and $N > M \cdot V$ where V is some upper bound on $v(y)$ so to be sure that no modular reduction will happen when performing the addition of the resulting shares. Concretely the sender sets $\gamma_i^y = (-u_i^2 + 1 \mod N, -v_i^2 + v(y) \mod N)$ for all $y \in Y$ and $\gamma_i^{\perp} = (-u_i^2 \mod N, -v_i^2 \mod N)$. After the protocol the receiver defines her shares u_i^1, v_i^1 to be the shares contained in her output of the PSM protocol, and then both parties add their shares locally to obtain secret sharing of the size of the intersection and of the sum of the values i.e., $U^1 = \sum_i u_i^1$, $V^1 = \sum_i v_i^1$, $U^2 = \sum_i u_i^2$, and $V^2 = \sum_i v_i^2$. Now the parties check if (U^1, U^2) is a sharing of 0 and, if not, they compute and reveal

[12] We recall that this technique yields to a protocol with a failure probability that is small (so the protocol it is valid from a practical point of view), but not negligible.

[13] We assume this only to simplify the protocol description, indeed our protocol can be easily instantiated when the two sets have different size.

the result of the computation $\frac{V^1+V^2}{U^1+U^2}$. Both these operations can be performed using efficient two-party protocols for comparison and division such as the one in [101, 29].

4.4.2 Threshold PSI

In this example we design a protocol $\Pi^t = (P_1^t, P_2^t)$ that securely computes the functionality $\mathscr{F}^t = (\mathscr{F}^t_{P_1^t}, \mathscr{F}^t_{P_2^t})$ where

$$\mathscr{F}^t_{P_1^t} \colon \{\{0,1\}^\lambda\}^M \times \{\{0,1\}^\lambda\}^M \longrightarrow \perp$$

and

$$\mathscr{F}^t_{P_2^t} \colon \{\{0,1\}^\lambda\}^M \times \{\{0,1\}^\lambda\}^M \longrightarrow \{\{0,1\}^\lambda\}^\star$$
$$(S_1, S_2) \longmapsto \begin{cases} S_1 \cap S_2 & \text{if } |S_1 \cap S_2| \geq t \\ \perp & \text{otherwise} \end{cases}$$

That is, the sender and the receiver have on input two sets, S_1 and S_2 respectively, and the receiver should only learn the intersection between these two sets if the size of the intersection is greater or equal than a fixed (public) threshold value t. In the case that the size of the intersection is smaller that t, then no information about S_1 is leaked to P_2^t and no information about S_2 is leaked to P_1^t. (This notion was recently considered in [49] in the context of privacy-preserving ride-sharing).

As in the previous example, the sender and the receiver run M executions of our protocol where the receiver inputs a different x_i from S_2 in each execution. The sender always inputs the same set S_1, and chooses the two outputs γ_i^0, γ_i^1 in the following way: γ_i^b is going to contain two parts, namely an arithmetic secret sharing of 1 if $x_i \in Y$ or 0 otherwise, as well as encryption of the same bit using a key k. The arithmetic secret sharing will be performed using a modulus larger than M, so that the arithmetic secret sharings can be added to compute a secret-sharing of the value $|S_1 \cap S_2|$ with the guarantee that no overflow will occur. Then, the sender and the receiver engage in a secure-two party computation of a function that outputs the key k to the receiver if and only if $|S_1 \cap S_2| > t$. Therefore, if the intersection is larger than the threshold now the receiver can decrypt the ciphertext part of the γ values and learn which elements belong to the intersection. The required 2PC is a simple comparison with a known value (the threshold is public) which can be efficiently performed using protocols such as [45, 70].

Chapter 5

Oblivious Querying of a Distributed Database

5.1 Introduction

We consider the setting where two companies, represented by two servers, hold a very large, secret-shared database. The data in this database itself is confidential, but also of potential value to other organisations. For this reason the companies wish to allow third parties to do certain kinds of approved computation on small samples of the data, in return for payment. However, which specific elements in the database are computed on is in itself private, i.e., can be considered a third party's private input, and thus may not be leaked to the companies holding the shared database. Furthermore, the data held in this shared database might be personal, and perhaps even unknown to either of the two hosting companies (since the entries are secret-shared), making it all the more vital to ensure that neither of the two hosting companies should learn which specific data elements are being computed on by a third party. In this chapter we show how to construct an efficient scheme for this kind of computation. In particular our focus is on reducing the communication and computation required by the client, to such an extent that the approach could be run as an app or in a web-browser. We achieve a solution that does not require asymmetric cryptography (outside of the underlying MPC computation) and communication and computation for the clients which is independent of the size of the database.

Even though we are agnostic to how the shared database came to be, one possible way could be as a combination of two different companies holding different info on the same entities. Specifically this means that we assume two companies (servers) A and B have a vertical sharing of a database where they both are aware of some primary key of all their data entries. For concreteness we might consider A and B to be companies with a large, common customer database, each holding different information on their customers. A could for example hold certain kinds of medical information and B could be a fitness tracking company holding information about customers' fitness habits. The companies determine their common customers, using an efficient private set intersection protocol [96], and afterwards create a sharing of their combined database on common customers. As a querying third party, we might consider a scientist or insurance company that wish to compute some statistics on a single person or subset of these people. If A and B are allowed to learn which data points to use for the computation, then solving this issue is trivial since a querying third party can simply request in plain what data points it wishes to use. This becomes non-trivial when we wish to hide the access pattern from the servers A and B. Since these learn the exact computation done and know the primary keys of each database entry, leaking the access patterns is highly undesirable. This is especially so if we consider a scenario

in which A and B provide a service to many different third parties; by correlating the access patterns of different third parties, the servers can gain a lot of information about which records are being queried. For this specific use case, it would of course also be necessary to provide some sort of external consent management to ensure that the querying third party does not illegitimately request computation of data subjects which have not given their consent. However, another relevant setting where consent is not needed, is one where both company A and B hold distinct information about certain specific geographic locations. For example A might be a surveying company holding information about the demographics of visitors of a certain place. B might hold info about commercial outlets in that area, e.g., profits, business peak times, etc. A third party might want to find the best location for an outlet catering for a specific demographic within a certain region.

In the following we will present a protocol for solving this issue assuming A and B do not collude and that no querying party colludes with A or B. However, any amount of querying third parties can collude without compromising the security of the scheme. The protocol only requires a PRF, a black-box MPC protocol (for the underlying MPC computation) and a PRP where specific entries can be queried efficiently, working over a small domain. The protocol provides semi-honest correctness, but maliciously secure privacy (assuming the underlying MPC protocol also provides this). We will implement and benchmark our scheme in deliverable 4.4.

5.2 Preliminaries

We let κ be the (computational) security parameter. Let N denote the amount of elements in the secret-shared database and M denote the maximum amount of bits in each element in the database. We let P be the amount of elements the client wishes to access and assume $P \ll N$. We assume access to an ideal MPC functionality and use the standard notation $[\![x]\!]$ to denote a closed value in the MPC scheme and assume standard operations on such values can be computed in MPC. That is, $[\![x]\!] \odot [\![y]\!] = [\![x \odot y]\!]$ where $\odot$ can be arithmetic operations.

We assume the setting of outsourced MPC [61], i.e., that a predetermined set of servers will handle the computation on behalf of a client. For simplicity we assume that the servers A and B holding the database are also the servers doing the MPC computation. It is trivial to achieve outsourced MPC in the semi-honest, non-colluding setting (with malicious privacy) by simply having the party outsourcing the computation construct an additive or XOR secret-sharing of its input and distribute the shares to the servers. The servers will use these shares as input and, in MPC, simply combine the shares and then do the actual computation. There are also several efficient approaches to achieve outsourced MPC in the fully malicious model only assuming black-box access to the underlying MPC scheme [57, 31]. Our scheme is agnostic to whether computation is over the binary field, a large field, or a ring. We assume for simplicity in the presentation that computation is over the binary field and use $\oplus$ to denote addition (XOR) and $\odot$ to denote multiplication (AND).

We also assume access to a pseudo-random function, sampled on a key $k \in \{0,1\}^\kappa$, working over input from $\mathbb{Z}_N$. Specifically, $\mathsf{PRF} : \{0,1\}^\kappa \times \mathbb{Z}_N \to \{0,1\}^*$, where we use the notation $\mathsf{PRF}_k(x) = y$ for $k \in \{0,1\}^\kappa$, $x \in \mathbb{Z}_N$ and $y \in \{0,1\}^*$. We assume PRF is efficiently computable and y is computationally indistinguishable from a random string to a probabilistic polynomial time (in κ) distinguisher for a randomly sampled key k.

We assume access to a pseudo-random permutation sampled through a seed $s \in S$ where the cardinality of S, $|S| \geq 2^\kappa$, and working over the domain of the positive integers less than some arbitrary value N. Specifically $\mathsf{PRP} : S \times \mathbb{Z}_N \to \mathbb{Z}_N$ is a mapping where, for each $s \in S$ it holds that PRP is a bijection on $\mathbb{Z}_N$, and we will use the notation $\mathsf{PRP}_s(x) = y$ for $s \in S$ and $x, y \in \mathbb{Z}_N$. We require that it is

efficient, i.e., independent of the domain size N, to compute the permuted position of a single element. Furthermore for any probabilistic polynomial time distinguisher the probability of distinguishing between PRP and a uniformly random sampled distributed permutation over $\mathbb{Z}_N$ is negligible in κ.

5.3 The Protocol

The overall idea of our protocol for outsourced computation on a secret-shared database is to have the querying third party (from now on denoted the *client*) obliviously learn a one-time padding of each *share* of *each* of the elements in the database it wishes to access. For each of these shares we then have the server, whose share the client did not learn, learn the corresponding one-time padded value. Then the client can input each of the paddings and each server input each of the padded values. The paddings can then be removed under MPC.

More concretely to allow a client to learn the shares of server A of some elements in the database the following protocol is executed: Server A picks a PRF key k and uses this to compute a one-time padding of its share of the database. After doing so, it picks a PRP seed s and permutes the entire padded database. A then sends the padded and permuted database to B and the PRF key and PRP seed to the client. Using k and s the client can compute the paddings and the permuted indexes of each of the shares it wishes to use. It can then send the permuted indexes in plain to server B since it does not know the permutation. This allows B to input the padded shares into an MPC computation and the client to input the paddings. This is symmetric for the share of server B. We formally describe this flow in Protocol 5 using a_i to denote the shares of element $i \in [N]$ in the database for server A. For this protocol description we denote the indexes the client wishes to obliviously retrieve the content for as $\iota_1, \ldots, \iota_P$ where it then holds that $\iota_i \in [N]$ for $i \in [P]$.

Protocol 5 Oblivious Querying of a Distributed Database

Execute the following twice, s.t. each server acts as A and B once.

Server $A(a_1, \ldots, a_N)$:
- $k \in_R \{0,1\}^\kappa$ ▷ PRF key
- $s \in_R S$ ▷ PRP key
- **for** $i \leftarrow 1$ **to** N **do**
  - $\bar{a}_i \leftarrow \mathsf{PRF}_k(i) \oplus a_i$ ▷ Compute padded value
  - $\hat{a}_i \leftarrow \bar{a}_{\mathsf{PRP}_s(i)}$ ▷ Compute permuted and padded value
- **return** (k, s) to the client
- **return** $(\hat{a}_1, \ldots, \hat{a}_N)$ to server B

Client$(\iota_1, \ldots, \iota_P)$:
- Receive (k, s) from server A
- **for** $i \leftarrow 1$ **to** P **do**
  - $\bar{c}_{\iota_i} \leftarrow \mathsf{PRF}_k(\iota_i)$ ▷ Compute padding
  - $\hat{\iota}_i \leftarrow \mathsf{PRP}_s(\iota_i)$ ▷ Permute padding
  - $\hat{c}_i \leftarrow \bar{c}_{\hat{\iota}_i}$
- Input $\hat{c}_i$ into the MPC to get $[\![\hat{c}_i]\!]$.
- **return** $(\hat{\iota}_1, \ldots, \hat{\iota}_P)$ to server B

Server $B()$:
- Receive $(\hat{a}_1, \ldots, \hat{a}_N)$ from server A
- Receive $(\hat{\iota}_1, \ldots, \hat{\iota}_P)$ from the client
- **for** $i \leftarrow 1$ **to** P **do**
  - $\hat{b}_i \leftarrow \hat{a}_{\hat{\iota}_i}$
- Input $\hat{b}_i$ into the MPC to get $[\![\hat{b}_i]\!]$.

MPC$([\![\hat{c}_1]\!], \ldots, [\![\hat{c}_P]\!], [\![\hat{b}_1]\!], \ldots, [\![\hat{b}_P]\!])$:
- **for** $i \leftarrow 1$ **to** P **do**
  - $[\![\hat{a}_i]\!] \leftarrow [\![\hat{b}_i]\!] \oplus [\![\hat{c}_i]\!]$ ▷ Reconstruct the shares of server A in MPC

5.3.1 Complexity and Efficiency

Before discussing the complexity and efficiency we note that the kind of pseudorandom permutation we require is not trivial to achieve efficiently using standard constructions. Specifically the issue is that standard constructions for pseudorandom permutations, such as AES, cannot be used as they work over a large domain. Fortunately we can turn to the field of *Format Preserving Encryption* for help. However, care must be taken to ensure that the specific construction used remains secure after the adversary has access to $\sqrt{(}N)$ queries of the PRP, which will be needed for our use-case. An option could for example be the scheme by Morris and Rogaway [77].

Communication We see that the communication between the servers is $2 \cdot N \cdot M$ bits, as each of them must send their padded shares to the other.

Server to client communication is limited to $O(\kappa)$ to retrieve the PRF key and PRP seed, plus the amount of communication to input the shares into the outsourced MPC protocol. This will be $2 \cdot P \cdot M$ bits.

Computation We note that we don't require the use of any asymmetric cryptographic primitives, but only very efficient symmetric primitives. The computation is however polynomial in $N \cdot M$ for each of the servers. Client-side computation however is only polynomial in $P \cdot M$.

Efficiency We are able to obtain communication and computation independent of database size for the client. The servers only require linear communication and computation in the database size, with small constants. However, the biggest issue regarding efficiency is that the servers must execute the protocol every time a (possibly) new client runs a query.

5.3.2 Security

We will not formally prove security of our protocol here, but simply sketch how a simulator would work when we assume an ideal functionality that takes XOR shares of the database as input from A and B and indexes as input from a client, and computes a public MPC computation using the XOR of the shares of values the client has picked. We assume a malicious server is able to do input substitution of its shares and also perform an additive attack on the other server's shares. Likewise, a malicious client is also allowed to introduce additive errors to its choice of input.

Corrupt server We first notice that a server A never learns anything since nothing is ever given as input to it from either server B or the client, outside of the ideal outsourced MPC functionality. In particular the simulator extracts the seed and key from A's communication to the client, and the padded database, from A's communication to server B. Based on this it can compute the input which will be A's share of the database. It can input this into the ideal functionality. We next look at server B. We see that B does not send anything to either the client or server A, thus it cannot do anything maliciously. Concretely the simulator will pick uniformly random values and sent these to B to act as the permuted and padded database. We note that this will be indistinguishable from what is sent in the real world by the security of the PRF function. On behalf of the client it will send random indexes which will be indistinguishable under the security of the PRP. The simulator then extracts the padded entries B gives as input to the outsourced MPC computation. Using its knowledge of the permutation and the PRF key in can compute the additive error B might make and use this as input on B's behalf to the ideal functionality.

Corrupt client We notice the client does not learn anything because it never sees anything based on the shares of the database. Concretely the simulator picks a uniformly random PRF key and seed for the PRP and sends these to the client. It then extracts the indexes selected by the client by computing the full permutation and looking up the values it sends to server B. The simulator will use these as input to the ideal functionality. Finally it computes the expected paddings to receive from the client using the PRF key and the extracted indexes. It then XORs this with the value the client actually sends to server B and computes the additive difference, which it inputs to the ideal functionality.

Multiple clients We note that a new permuted and padded database must be exchanged between the servers for each new client that wish to use the system. This is necessary since re-using the same permutation across clients would leak whether two clients accessed the same elements or not. Furthermore, any client colluding with a server could break the privacy of the other server which is a huge issue. Fixing this issue is not trivial and is something we will investigate in the future.

Chapter 6

Private Training of Decision Trees with Continuous Attributes

Here we give a brief overview of work on the secure training of decision trees. This work is to be submitted to the 41st IEEE Symposium on Security and Privacy (S&P 2020)[1]. To comply with the requirements for original and unpublished work we only provide an abstract here and refer to the paper-to-appear for the full description and details.

In this work we show how to train a decision tree securely from a database that is secret-shared among multiple mutually distrustful parties. We consider data with continuous (i.e. real-valued) attributes, and develop a secure version of a learning algorithm similar to the C4.5 or CART algorithms. Previous work only focused on decision tree learning with discrete attributes [36].

Our protocol is developed in the client-server model, where the data owners secret-share their data towards a given set of servers, whose majority we assume to be honest. These servers will run the actual computation. For the protocol, we do not make very particular assumptions about the underlying MPC engine, but instead assume general primitives like linear secret-sharing, multiplication, sampling secret-shared random bits, etc.

We assume a very general scenario, in which both the input and the output of the secure algorithm are secret-shared. This allows the training algorithm to be used in a fully oblivious pipeline—for example, the secret-shared output might consequently be used to provide secure inference to other clients.

Techniques We exploit the fact that even if we allow the data to have continuous values, which a priori might require fixed or floating point representations, the output of the tree learning algorithm only depends on the relative ordering of the data. By obliviously sorting the data we reduce the number of comparisons needed per node to $O(N \log^2 N)$ from the naive $O(N^2)$, where N is the number of training records in the dataset, thus making the algorithm feasible for larger datasets. This does however introduce a problem when duplicate values occur in the dataset, but we manage to overcome this problem with a relatively cheap subprotocol.

We compute decision trees up to a fixed depth. Our protocol is split in three phases: first, we sort the data on each attribute. This is the most expensive single step for larger datasets, but only needs to be done once. Second, we compute each inner (non-leaf) node in the tree. This involves permuting values, computing Gini indices, and taking an argmax to find the best attribute and splitting point. Third and last, we compute the values for the leaf nodes.

[1] https://www.ieee-security.org/TC/SP2020/index.html

We need to not only sort data, but we also need to obliviously apply the resulting permutation to a vector. For this reason we use (practically efficient) sorting networks, which have complexity $O(N \log^2 N)$ and depth $O(\log^2 N)$. Since the resulting permutation needs to be applied once for every node in the tree, we show how to convert the resulting network into a permutation network with $O(N \log N)$ gates and $O(\log N)$ depth.

Implementation We implement our algorithm in the MP-SPDZ framework.[2] This framework has many different MPC protocols under the hood. Since secure comparisons are the bottleneck in our protocol, we choose to focus on arithmetic modulo 2^{64}, which has been shown to lead to faster comparisons [32]. Our benchmarks demonstrate that our approach scales to practical sizes of datasets. We also apply our training algorithm to a real dataset from a large scale medical use case.

[2] https://github.com/data61/MP-SPDZ

Bibliography

[1] A. A. Albert. Symmetric and alternate matrices in an arbitrary field, I. *Transactions of the American Mathematical Society*, 43(3):386–436, 1938.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 20–29. ACM, 1996.

[3] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 535–548, 2013.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 673–701, 2015.

[5] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 691–702, 2011.

[6] R. B. Bapat, K. P. S. Bhaskara Rao, and K. Manjunatha Prasad. Generalized inverses over integral domains. *Linear Algebra and its Applications*, 140:181–196, 1990.

[7] 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.

[8] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In José D. P. Rolim and Salil P. Vadhan, editors, *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Cambridge, MA, USA, September 13-15, 2002, Proceedings*, volume 2483 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002.

[9] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513, 1990.

[10] A. Ben-Israel. A volume associated with $m \times n$ matrices. *Linear Algebra and its Applications*, 167:87–111, 1992.

[11] A. Ben-Israel and T. N. E. Greville. *Generalized Inverses - Theory and Applications*. CMS Books in Mathematics. Springer, 2003.

[12] Kevin S. Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. On synopses for distinct-value estimation under multiset operations. In Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 199–210. ACM, 2007.

[13] F. Blom, N. J. Bouman, Berry Schoenmakers, and N. de Vreede. Efficient secure ridge regression from randomized Gaussian elimination. Cryptology ePrint Archive, Report 2019/773, 2019.

[14] D. Bogdanov, L. Kamm, S. Laur, and V. Sokk. Rmind: A tool for cryptographically secure statistical analysis. *IEEE Trans. Dependable Sec. Comput.*, 15(3):481–495, 2018.

[15] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multiparty computation for data mining applications. *Int. J. Inf. Secur.*, 11(6):403–418, November 2012.

[16] A. Borodin, J. von zur Gathen, and J. Hopcroft. Fast parallel matrix and GCD computations. *Information and Control*, 52(3):241–256, 1982.

[17] T. L. Boullion and P. L. Odell. *Generalized Inverse Matrices*. Wiley, 1971.

[18] N. J. Bouman and N. de Vreede. New protocols for secure linear algebra: Pivoting-free elimination and fast block-recursive matrix decomposition. Cryptology ePrint Archive, Report 2018/703, 2018.

[19] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1243–1255. ACM, 2017.

[20] L. Chen, W. Eberly, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Efficient matrix preconditioners for black box linear algebra. *Linear Algebra Its Appl.*, 343–344:119–146, 2002.

[21] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. *IACR Cryptology ePrint Archive*, 1998:3, 1998. Appeared in the THEORY OF CRYPTOGRAPHY LIBRARY.

[22] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, pages 464–482, 2018.

[23] R. J. F. Cramer and I. B. Damgård. Secure distributed linear algebra in a constant number of rounds. In *Proc. CRYPTO 2001, Santa Barbara, USA*, pages 119–136. Springer, 2001.

[24] R. J. F. Cramer, I. B. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Proc. TCC 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, 2005.

[25] R. J. F. Cramer, I. B. Damgård, and J. B. Nielsen. *Secure multiparty computation and secret sharing: An information theoretic approach*. Cambridge University Press, 2015.

[26] R. J. F. Cramer, E. Kiltz, and C. Padró. A note on secure computation of the Moore–Penrose pseudoinverse and its application to secure linear algebra. In *Proc. CRYPTO 2007*, volume 4622, pages 613–630. Springer, 2007.

[27] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers*, pages 143–159, 2010.

[28] Emiliano De Cristofaro and Gene Tsudik. Experimenting with fast private set intersection. In *Trust and Trustworthy Computing - 5th International Conference, TRUST 2012, Vienna, Austria, June 13-15, 2012. Proceedings*, pages 55–73, 2012.

[29] Morten Dahl, Chao Ning, and Tomas Toft. On secure two-party integer division. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, pages 164–178, 2012.

[30] I. B. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Proc. TCC 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, 2006.

[31] Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential benchmarking based on multiparty computation. In Jens Grossklags and Bart Preneel, editors, *FC 2016: 20th International Conference on Financial Cryptography and Data Security*, volume 9603 of *Lecture Notes in Computer Science*, pages 169–187, Christ Church, Barbados, February 22–26, 2016. Springer, Heidelberg, Germany.

[32] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1102–1120. IEEE, 2019.

[33] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 1–18, 2013.

[34] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer Berlin Heidelberg, 2012.

[35] Sebastiaan de Hoogh. *Design of large scale applications of secure multiparty computation: secure linear programming*. PhD thesis, Eindhoven University of Technology, 2012.

[36] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. Practical Secure Decision Tree Learning in a Teletreatment Application. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 179–194, 2014.

[37] Sebastiaan de Hoogh, Berry Schoenmakers, and Meilof Veeningen. *Universally Verifiable Outsourcing and Application to Linear Programming*, volume 13 of *Cryptology and Information Security Series*, chapter 10. IOS Press, 2015.

[38] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, volume 2832 of *Lecture Notes in Computer Science*, pages 605–617. Springer, 2003.

[39] W. Eberly and E. Kaltofen. On randomized Lanczos algorithms. In *Proc. ISSAC '97*, pages 176–183. ACM, 1997.

[40] Cristian Estan, George Varghese, and Michael E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006.

[41] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. *IACR Cryptology ePrint Archive*, 2018:238, 2018.

[42] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[43] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 303–324, 2005.

[44] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 1–19, 2004.

[45] Juan A. Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, pages 330–342, 2007.

[46] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Privacy-preserving distributed linear regression on high-dimensional data. *PoPETs*, 2017(4):345–364, 2017.

[47] 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.

[48] T. Greville. Note on the generalized inverse of a matrix product. *SIAM Review*, 8(4):518–521, 1966.

[49] Per Hallgren, Claudio Orlandi, and Andrei Sabelfeld. Privatepool: Privacy-preserving ridesharing. In *IEEE 30th Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 276–291, 2017.

[50] R. E. Hartwig. The reverse order law revisited. *Linear Algebra and its Applications*, 76:241–246, 1986.

[51] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, pages 155–175, 2008.

[52] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*, pages 175–203, 2017.

[53] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[54] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Report 2017/738, 2017. http://eprint.iacr.org/2017/738.

[55] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 145–161, 2003.

[56] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 575–594, 2007.

[57] Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. *IACR Cryptology ePrint Archive*, 2016o:37, 2016.

[58] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, pages 418–435, 2010.

[59] E. Kaltofen and A. Lobo. On rank properties of Toeplitz matrices over finite fields. In *Proc. ISSAC '96*, pages 241–249. ACM, 1996.

[60] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Proc. 9th Int. Symp. AAECC*, volume 539 of *LNCS*, pages 29–38. Springer, 1991.

[61] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. http://eprint.iacr.org/2011/272.

[62] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 195–215, 2014.

[63] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In Jan Paredaens and Dirk Van Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 41–52. ACM, 2010.

[64] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 724–741, 2015.

[65] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. *IACR Cryptology ePrint Archive*, 2016:505, 2016.

[66] E. Kiltz, P. Mohassel, E. Weinreb, and M. Franklin. Secure linear algebra using linearly recurrent sequences. In *Proc. TCC 2007*, volume 4392 of *LNCS*, pages 291–310. Springer, 2007.

[67] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 54–70, 2013.

[68] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 818–829, 2016.

[69] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[70] Helger Lipmaa and Tomas Toft. Secure equality and greater-than tests with sublinear online complexity. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, pages 645–656, 2013.

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

[72] G. Marsaglia and G. P. H. Styan. Equalities and inequalities for ranks of matrices. *Linear and Multilinear Algebra*, 2(3):269–292, 1974.

[73] G. Marsaglia and G. P. H. Styan. Rank conditions for generalized inverses of partitioned matrices. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 437–442, 1974.

[74] Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*, pages 134–137, 1986.

[75] P. Mohassel and E. Weinreb. Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries. In *Proc. CRYPTO 2008*, volume 5157 of *LNCS*, pages 481–496. Springer, 2008.

[76] Payman Mohassel and Salman Niksefat. Oblivious decision programs from oblivious transfer: Efficient reductions. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, pages 269–284, 2012.

[77] Ben Morris and Phillip Rogaway. Sometimes-recurse shuffle - almost-random permutations in logarithmic expected time. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 311–326, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[78] Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, October 1978.

[79] K. Mulmuley. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica*, 7(1):101–104, 1987.

[80] J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

[81] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding P2P bots with structured graph analysis. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 95–110, 2010.

[82] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *Proc. 2013 IEEE Symp. on Security and Privacy*, pages 334–348. IEEE, 2013.

[83] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Proc. PKC 2007*, volume 4450 of *LNCS*, pages 343–360. Springer, 2007.

[84] K. Nissim and E. Weinreb. Communication efficient secure linear algebra. In *Proc. TCC 2006*, volume 3876 of *LNCS*, pages 522–541. Springer, 2006.

[85] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-n OT extension with application to private set intersection. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, pages 381–396, 2017.

[86] Elena Pagnin, Gunnar Gunnarsson, Pedram Talebi, Claudio Orlandi, and Andrei Sabelfeld. Toppool: Time-aware optimized privacy-preserving ridesharing. *PoPETs*, 2019(4):93–111, 2019.

[87] Martin H. Pearl. Generalized inverses of matrices with entries taken from an arbitrary field. *Linear Algebra and its Applications*, 1(4):571–587, 1968.

[88] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pages 515–530, 2015.

[89] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 125–157. Springer, 2018.

[90] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 797–812, 2014.

[91] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. Cryptology ePrint Archive, Report 2016/930, 2016. http://eprint.iacr.org/2016/930.

[92] C. R. Rao. *Linear Statistical Inference and its Applications*. Wiley, 1973.

[93] C. R. Rao and S. K. Mitra. *Generalized inverse of matrices and its applications*. Wiley, 1971.

[94] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, pages 235–259, 2017.

[95] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1229–1242. ACM, 2017.

[96] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17: 24th Conference on Computer and Communications Security*, pages 1229–1242, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[97] C. A. Rohde. Generalized inverses of partitioned matrices. *Journal of the Society for Industrial and Applied Mathematics*, 13(4):1033–1035, 1965.

[98] Adi Shamir. On the power of commutativity in cryptography. In *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, pages 582–595, 1980.

[99] J. Springer. Die exakte Berechnung der Moore–Penrose-Inversen einer Matrix durch Residuenarithmetik. *Zeitschrift für Angewandte Mathematik und Mechanik*, 63(3):203–210, 1983.

[100] Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N. Asokan. The circle game: Scalable private membership test using trusted hardware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 31–44, 2017.

[101] Tomas Toft et al. Primitives and applications for multi-party computation. *PhD Thesis, University of Aarhus, Denmark*, 2007.

[102] P. S. Wang. A p-adic algorithm for univariate partial fractions. In *Proc. SYMSAC '81*, pages 212–217. ACM, 1981.

[103] 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, 1982.