Unconditionally Secure, Universally Composable Privacy Preserving Linear Algebra

Bernardo David[1] Rafael Dowsley[2] Jeroen van de Graaf[3]
Davidson Marques[3] Anderson C. A. Nascimento[4] Adriana C. B. Pinto[5]

December 2014

[1] Department of Computer Science, Aarhus University
Åbogade 34, 8200 Aarhus N, Denmark
`bernardo@cs.au.dk`

[2] Institute of Theoretical Informatics, Karlsruhe Institute of Technology
Am Fasanengarten 5, Geb. 50.34, 76131 Karlsruhe, Germany
`rafael.dowsley@kit.edu`

[3] Department of Computer Science, Federal University of Minas Gerais
Av. Antônio Carlos 6627, 31270-901, Belo Horizonte - MG, Brazil
`jvdg@lcc.ufmg.br`; `rodrigue@dcc.ufmg.br`

[4] Center for Data Science, Institute of Technology, University of Washington Tacoma
1900 Commerce Street, Tacoma, WA 98402-3100, USA
`andclay@uw.edu`

[5] Department of Electrical Engineering, University of Brasília
Campus Darcy Ribeiro, 70910-900, Brasília - DF, Brazil
`adrianacbp@redes.unb.br`

Abstract

Linear algebra operations on private distributed data are frequently required in several practical scenarios (*e.g.* statistical analysis and privacy preserving databases). We present universally composable two-party protocols to compute inner products, determinants, eigenvalues and eigenvectors. These protocols are built for a two-party scenario where the inputs are provided by mutually distrustful parties. After execution, the protocols yield the results of the intended operation while preserving the privacy of their inputs. Universal composability is obtained in the trusted initializer model, ensuring information theoretical security under arbitrary protocol composition in complex environments. Furthermore, our protocols are computationally efficient since they only require field multiplication and addition operations.

Keywords: UC security, Trusted Initializer Model, Linear Algebra.

1 Introduction

Secure computation is an important recurrent topic in modern cryptography that deals with the problem of two or more mutually untrusting parties that want to make computations over their data without revealing their inputs to each other. General solutions for this seemingly impossible problem are known for both two-party and multi-party cases with different security guarantees [17, 63, 35, 44, 20]. These solutions mainly address the question of evaluating boolean and arithmetic circuits in two-party and multi-party settings with computational or unconditional security.

Protocols for secure two-party computation of boolean and arithmetic circuits constitute a nice general solution to the problem of secure computation. However, these protocols pay a high price in terms of efficiency in order to achieve generality. Usually, tailoring a protocol to a specific operation achieves better computational and communication efficiency in comparison to computing the circuit that executes such operation through a general secure two-party computation protocol. Protocols for different specific purposes have been proposed, such as protocols for scalar products [34], means [45], statistics [12], equality [21], comparison [21] and exponentiations [21].

Linear algebra operations are needed in several natural applications of secure two-party computation, such as data mining, statistics and benchmarking. For example, statistical analysis tasks involving linear algebra arise in application such as privacy preserving benchmarking, an application proposed in current literature [42, 60] where two companies wish to compare their performance without revealing to each other sensitive financial and strategical data. Another interesting application is the use of linear programming to perform privacy preserving supply-chain management [43, 56], where a supplier and a costumer have to agree on the best dates for manufacturing and delivery of supplies without revealing sensitive strategical information.

In this paper we consider secure two-party computation of vector inner products, matrix multiplications, determinants, eigenvalues and eigenvectors. Such operations can be employed in different scenarios to solve linear systems, compute the correlation between datasets and perform linear programming tasks. We investigate protocols tailored to each specific linear algebra operation that we wish to compute. As opposed to general approaches to practical secure computation such as [10, 11, 36, 41], we wish to obtain protocols that achieve high efficiency, security and parallelism for specific operations.

Even though many classical two-party computation protocols have been proven secure in a stand-alone setting, it is important to analyze the security of cryptographic protocols under scenarios where many copies of the protocol are executed in parallel. The real/ideal simulation paradigm and universal composability (UC) [13] provide widely accepted security notions for analyzing protocols under sequential and arbitrary composition, respectively. As a consequence of being secure under arbitrary composition, UC secure protocols and constructions can be used as building blocks for more complex applications and other cryptographic protocols.

It is well-known that two-party computation is only possible with some additional assumption, either about the computational resources available to the parties (e.g. polynomial time, bounded memory), or some setup assumption (e.g. a noisy channel, oblivious transfer, the help of some trusted party). In the UC framework, the scenario is even more restricted, as it is known that protocols for two-party and multi-party functionalities cannot be build without setup assumptions [14, 15]. Non-trivial UC-secure functionalities can be implemented considering setup assumptions such as: common reference string [14, 15, 55], public-key infrastructure [3], trusted initializer [28], random oracle model [37], noisy-channels [27, 29], signature cards [38] and tamper-proof hardware [40, 49, 25]. Pre-distribution of correlated randomness (*e.g.* precomputed operations) by a trusted initializer constitutes an interesting setup assumption for obtaining UC secure two-party protocols and is the one considered in this paper.

Trusted Initializers were introduced as part of the commodity based cryptography model by Beaver [6, 7], which is inspired by client-server architectures. In the commodity based cryptography model, a third party referred to as the trusted initializer (TI) correctly precomputes a number of operations over random inputs obtaining correlated randomness. This predistributed correlated randomness is then used by the users to compute the desired function over their own private inputs (by "derandomizing" the pre-distributed data). The main goal of this model is achieving very high security guarantees (*i.e.* unconditional security) while remaining efficient (*i.e.* all the heavy computation is offloaded to the trusted initializer while the users only have to execute simple computational operations).

1.1 Related Work

There is a long line of work in obtaining protocols for secure two-party or multiparty computation of any function, starting with the seminal results by Yao [63] in the two-party case and Chaum *et al.* [17] and Goldreich *et al.* [35] in the multiparty case. Currently there are a number of approaches to performing efficient secure multiparty computation in different scenarios. Recent efforts include optimizing Yao's garbled circuits technique [63] for two-party computation [32, 48, 47], evaluating RAM programs [1, 41, 62], optimizing oblivious transfer based protocols [52, 59, 2] and constructing protocols in the preprocessing model [23, 22, 24]. In this paper we concentrate on the commodity based cryptography approach, which mostly resembles the preprocessing model. Hence, we focus our analysis of previous literature on these two models.

A naive approach to obtain secure two-party computation for linear algebra operations based on a trusted initializer consists in using general protocols for secure two-party computation. Oblivious Transfer [31] is known to be complete for any two-party functionality [44] and protocols for oblivious transfer in the commodity based cryptography model were introduced in [6, 5]. Hence, in theory, any linear algebra operation can be computed with a trusted initializer. However, the general protocol described in [44] requires a high number of rounds and oblivious transfer calls.

It is also possible to perform secure computation of linear algebra operations using efficient protocols for general secure two-party or multiparty computation in the preprocessing model such as [4, 8, 23, 22, 24]. In this model, a preprocessing phase takes places before the actual computation on real inputs in order to distribute correlated data to the parties, resulting in very efficient online phases (*i.e.* actual computation on inputs). This model is closely related to commodity based cryptography since it also relies on pre-distribution of correlated data that represents precomputed operations.

Even though the currently most efficient protocols for general multiparty computation achieve good performance, they still require a large number of rounds and complex operations in comparison to protocols tailored to specific functionalities. In general, these protocols require programs to be represented by binary circuits [52, 32, 48, 47], arithmetic circuits [8, 23, 22] or a combinations of both [24]. These circuits then have to be evaluated gate by gate, which introduces an overhead that grows with the number of gates. Moreover, general purpose secure multiparty protocols secure against malicious adversaries (that can arbitrarily deviate from the protocol) need to employ mechanisms that introduce extra overhead, such as message authentication codes [8, 23, 22] and complicated circuit validation techniques [32, 48, 47]. The protocols that achieve the best concrete performance [59, 24] are only secure against semi-honest adversaries, meaning that they are only secure as long as the adversaries do not deviate from the protocol steps.

In order to avoid the overhead inherent to general purpose secure multiparty computation protocols, another line of work has been focusing on designing secure computation protocols for specific functionalities. Cramer and Damgård introduced the first constant-round unconditionally secure protocols for linear algebra in [18]. These protocols are stated in the multiparty setting assuming access to standard constant-round protocols for generation of random shares, addition and multiplication of shares. Departing from these building blocks, they introduce protocols for computing the determinant, characteristic polynomial and rank of matrices as well as solving linear systems of equations. The results of [18] are subsequently improved in [19] through a technique for securely computing the Moore-Penrose Pseudoinverse based on the same basic building blocks of [18] that allows securely solving linear systems of equations with better complexity (for a system of m linear equations with n variables this result requires $m^4 + n^2m$ secure multiplications instead of n^5).

Similar problems were also studied in the two-party case with computational security in [53], where protocols based on public-key homomorphic encryption and Yao's garbled circuits were proposed. Specifically, they introduce protocols for securely computing linear subspace intersection and oblivious gaussian elimination, leading to protocols for solving linear systems of equations. These protocols only achieve round complexity of $O(n^{0.275})$ but reduce communication complexity to $O(n^2)$. These results are improved in [46], which presents protocols for deciding if a singularity and computing the minimal polynomial of a encrypted matrices with round complexity $O(\log n)$ and communication complexity $O(n^2)$. These protocols are applied to securely computing rank and determinant of matrices, and to solving systems of linear equations with similar complexities.

Another proposal for tailor-made secure two-party computation protocols for linear algebra operations was put forth in the Ph.D. thesis of Du [30]. His thesis demonstrates the importance of practical secure two-party computation protocols for specific tasks and proposes several protocols for realizing such tasks. However, the protocols proposed in [30] lack a rigorous security analysis and, in fact, there are fatal flaws in most of these protocols as noted in [34, 45]. Posterior works provided secure protocols for some of the tasks proposed by Du: protocols for scalar multiplication [34] (based on homomorphic encryption) and means [45] (based on building blocks that can be obtained with unconditional security). A conference version of part of this work [26] presented a protocol for computing the vector inner product in the trusted initializer model. Recently, another protocol for computing the vector inner product in the trusted initializer model was proposed in [39], however it deals with a slightly different definition of the inner product functionality than the one considered in this paper, and requires more rounds and operations.

The commodity based cryptography model was introduced by Beaver [6]. In this model, commitment and oblivious transfer protocols were presented in [6, 58], protocols for some secure two-party computations in [50], an oblivious polynomial evaluation protocol in [61] and a verifiable secret sharing scheme in [28].

1.2 Our Contributions

In this paper, we introduce two-party protocols for privacy preserving computation of linear algebra operations in a finite field over inputs provided by both parties. Namely, we first expand on the protocols for computing vector inner products, matrix multiplication and solving systems of linear equations that we introduced in a conference version of this paper [26], providing full descriptions and security proofs. We then use these protocols as building blocks for obtaining new protocols for determinants, eigenvalues and eigenvectors. All of our protocols achieve statistical security against static malicious adversaries in the UC model using a trusted initializer as a setup assumption. We present protocols for securely computing the following operations:

- **Inner Product:** Round-optimal (2 rounds) with communication complexity of $2k + 1$ elements for k elements long vectors.
- **Matrix Multiplication:** For multiplying $m \times k$ and $k \times p$ matrices, this protocol is constructed as mp (parallel) invocations of the Inner Product protocol and thus also achieves 2 rounds. The communication complexity is of $mp(2k + 1)$ elements.
- **Solving Linear Systems of Equations:** For linear systems of n equations with n variables this protocol requires 5 rounds (including 2 invocations of the matrix multiplication protocol that run in parallel). The communication complexity is of $(2n^3+n^2)+n^2+(2n^2+n)+(n^2+n)+n = 2n^3+5n^2+3n$ elements.
- **Determinant:** For $n \times n$ matrices, this protocol requires 5 rounds (including an invocation of the matrix multiplication protocol). The communication complexity is of $(2n^3 + n^2) + n^2 + n^2 + n = 2n^3 + 3n^2 + n$ elements.
- **Eigenvalue:** For $n \times n$ matrices, this protocol requires 4 rounds (including an invocation of the matrix multiplication protocol). The communication complexity is of $(2n^3 + n^2) + n^2 + n^2 = 2n^3 + 3n^2$ elements.
- **Eigenvector:** For $n \times n$ matrices, this protocol requires 5 rounds (including an invocation of the matrix multiplication protocol). The communication complexity is of $(2n^3 + n^2) + n^2 + n^2 + n = 2n^3 + 3n^2 + n$ elements.

These protocols enjoy the following characteristics:

- **Privacy Preserving Linear Algebra Operations:** Protocols for secure and private two-party computation of vector inner products, matrix multiplications, determinants, eigenvalues and eigenvectors.
- **Universal Composability:** Our protocols are proven secure in the universal composability framework, thus retaining security in complex and realistic environments.
- **Unconditional Security:** Our protocols remain secure even against computationally unbounded adversaries and do not rely on any computational assumptions (under the setup assumption that correlated randomness exists).
- **Efficiency:** Our protocols only require addition and multiplication over finite fields, which can be efficiently parallelized and computed in both massive data and resource constrained environments.

We construct protocols for each operation directly, eliminating the need of expressing each of them in terms of boolean or arithmetic circuits, what consequently eliminates the extra overhead in computational, communication and round complexities incurred by general purpose two-party computation protocols. Our protocols are *universally composable* and achieve security against *malicious users*, in contrast to some of the current most efficient protocols for general purpose two-party computation [59, 24], which only achieve security against semi-honest adversaries. We prove that our constructions achieve *unconditional security*, meaning that we do not rely on computational assumptions such as in [32, 48, 47]. Moreover, our protocols do not require message authentication codes or any extra overhead to achieve security against malicious adversaries, which is the case in the online phases of [8, 52, 23, 22].

The fundamental building block of the protocols introduced in this paper is a protocol for secure two-party computation of distributed vector inner products, which is used in a straightforward construction of distributed matrix multiplication. These protocols were both introduced in our conference paper [26]. For these operations, we consider a scenario where two mutually untrusting parties A and B give private vectors (or matrices) as inputs and wish to compute distributed operations without revealing their inputs to each other. By *distributed* operations we mean that both the inputs *and the outputs* of a given operation are distributed among the two parties that compute it, *i.e.* each party contributes part of the operations' input and receives a share of the operations' output.

The matrix multiplication protocol is then used as a building block to obtain new protocols for secure two party computation of determinants, eigenvalues and eigenvectors over sums of matrices. In this case, we consider that each party inputs a matrix and only one of the parties obtains the result of the operation over the sum of both inputs. The security property guarantees that none of the parties are able to learn each other inputs.

Being based on a trusted initializer that precomputes operations, our protocols achieve high computational and communication efficiency. The number of required rounds is significantly smaller than in general two-party computation protocols since our protocols are tailored for specific functionalities. Both the TI and the parties involved in the protocol are only required to execute addition and multiplication over a finite field. Such operations are simple enough to be executed in resource constrained environments (*e.g.* embedded computers) while being easily parallelizable for simultaneous execution of several protocol instances in large scale environments (*e.g.* big data applications). Furthermore, our protocols are not based on specific computational assumptions and achieve unconditional security, meaning that they resist attacks from computationally unbounded adversaries.

In comparison to [18, 19] our protocol for solving linear systems of n equations with n variables achieve the same asymptotic round complexity but lower communication complexity as our protocol's communication complexity is of the order of $O(n^3)$ while their protocols require $\Omega(n^4)$ secure multiplications. Our protocols still do not achieve the same communication complexity of $O(n^2)$ as protocols based on computational assumptions [53, 34, 46] but achieve better round complexity without the need for computational assumptions (as do the results of [18, 19]).

We evaluate the concrete efficiency of our protocols by analyzing the concrete performance of prototype implementations taking inputs of different sizes. We also analyze an optimized implementation of the inner product protocol, upon which the other protocols are based. Our implementation guarantees security in distributed linear algebra operations only incurring a small extra computational cost in relation to a trivial implementation that does not provide any security at all.

1.3 Organization

In Section 2, we establish notation and introduce definitions that will be used throughout the paper. In Section 3, we revisit the protocols for computing vector inner products and matrix multiplication that we previously introduced in [26], providing full descriptions and security proofs. In Setion 4, we revisit the protocol for solving linear equation systems that we previously introduced in [26], providing full descriptions and security proofs. In Section 5, we introduce a protocol for computing determinants and analyse its security. In Section 6, we introduce a protocol for computing eigenvalues and analyse its security. In Section 7, we introduce a protocol for computing eigenvectors and analyse its security. In Section 8 we analyse the complexity and concrete efficiency of our protocols, presenting concrete performance analysis of prototype implementations.

2 Preliminaries

In this section we establish notation and introduce definitions that will be used throughout the paper.

2.1 Notation

Hereupon, we will denote by $x \xleftarrow{\$} D$ an uniformly random choice of element x over the domain D; by $\oplus$ a bit-wise exclusive OR of strings; and by $a \,\|\, b$ the concatenation of string a with string b. All logarithms are to the base 2. For a probabilistic polynomial-time (PPT) machine A, we use $\mathsf{coins(A)}$ to denote the distribution of the internal randomness of A and $a \xleftarrow{\$} A$ to denote running the machine A with internal randomness distributed according to $\mathsf{coins(A)}$ and obtaining the output a.

In the following, we denote by $\mathbb{F}_q$ the finite field of order q, by $\mathbb{F}_q^n$ the space of all n-tuples of elements of $\mathbb{F}_q$. $\mathbb{F}_q^{m\times n}$ represents the space of all $m \times n$ matrices with elements belonging to $\mathbb{F}_q$, while $SL(\mathbb{F}_q, n)$ represents the set of all non-singular $n \times n$ matrices with elements belonging to $\mathbb{F}_q$.

Two sequences X_λ and Y_λ of random variables are said to be *computationally indistinguishable*, denoted by $X \overset{c}{\approx} Y$, if there exists a negligible function $\epsilon(\cdot)$ such that for every $\lambda \in \mathbb{N}$ and for every non-uniform PPT distinguisher D it holds that

$$|\Pr[\,D(X_\lambda) = 1\,] - \Pr[\,D(Y_\lambda) = 1\,]| < \epsilon(\lambda).$$

If the previous inequality holds also for all computationally unbounded distinguishers, then the sequences are *statistically close*, denote $X \overset{s}{\approx} Y$.

2.2 Commodity Based Cryptography

In this section, we briefly discuss the commodity based cryptography model introduced by Beaver [6, 7]. This model was inspired by the client-server distributed computation model, where a powerful server performs complex tasks on behalf of a client. It is an efficient alternative for obtaining secure multi-party computation.

In this model, a trusted initializer precomputes certain operations that are then used by individual parties to execute a given protocol. The parties access such operations by requesting correlated, pre-distributed values (the commodities) to the TI before they start executing the protocol itself. It is possible to obtain very efficient protocols in this model, since most of the required complex operations can be delegated to the TI and then pre-distributed to the parties. The trusted initializer is always assumed to be honest. Notably, this model has been used to construct commitments [58, 9, 50], oblivious transfer [6, 5], verifiable secret sharing [51, 28] and oblivious polynomial evaluation [61].

Notice that the TI has no access to the parties' secret inputs nor does it receive any information from the parties. The only communication required between the TI and the parties occurs during a setup phase, when the TI pre-distributes information. Among the main advantages of this model is the high computational efficiency that arises from the fact that the parties running the protocol are not required to compute complex operations but only derandomize the precomputed instances to match their own inputs. Moreover, since the operations are computed by a trusted initializer and not by interaction among the parties, they generally yield perfect results regardless of any attacks. In other words, it is possible to obtain statistically secure protocols with low computational and communication complexity.

2.3 The Universal Composability Framework

In this section, we present a brief discussion of the Universal Composability framework. We refer the readers to [13, 16] for further details. The main goal of the Universal Composability framework [13] is to analyze the security of cryptographic protocols under arbitrary composition. In other words, this framework takes into consideration scenarios where many copies of a protocol are executed concurrently with themselves and other protocols, such as the Internet. The composition theorem presented in [13] ensures that any protocol proven to be secure under the UC framework can also be securely composed with copies of itself and other protocols. Apart from guaranteeing security in a realistic scenario, the UC framework also enables the utilization of UC-secure protocols as building blocks for complex applications.

In the UC framework we consider a set of parties interacting with each other, an adversary $\mathcal{A}$ that can corrupt the parties, and an *environment* $\mathcal{Z}$. The environment is responsible for providing the inputs for the parties and $\mathcal{A}$, and receiving their outputs. The adversary $\mathcal{A}$ may choose to corrupt a set of parties, thus gaining control over their communication channels and computation. All these entities are modeled as Interactive Turing Machines.

The main idea behind the UC framework is that $\mathcal{Z}$ represents all activity external to the current execution of the protocol. In order to prove the security of a specific protocol, one must define an ideal version of the functionality that the protocol is supposed to realize and an ideal adversary $\mathcal{S}$. Then, we have to show that no $\mathcal{Z}$ can distinguish between an execution of the specific protocol implementation with the parties and $\mathcal{A}$, and an ideal execution with the parties and $\mathcal{S}$.

The ideal version of the protocol is called the ideal functionality $\mathcal{F}$ and it does exactly what the protocol should do in a black box manner. In other words, given the inputs, the ideal functionality follows the primitive specification and returns the output as specified. However, the functionality must deal with the actions of corrupted parties, such as invalid inputs and deviations from the protocol. Some interesting points about $\mathcal{F}$ are: the communication between the parties and $\mathcal{F}$ are made by writing on their input and output tapes; $\mathcal{S}$ has no access to the contents of messages sent between the parties and $\mathcal{F}$ unless the party is corrupted; $\mathcal{Z}$ cannot see the messages sent between the parties and $\mathcal{F}$ (and cannot see the messages sent between the parties in the real protocol execution).

The ideal adversary $\mathcal{S}$, or simulator, is normally designed to act in the same way as the adversary $\mathcal{A}$ in the interaction with the real protocol. It means that every attack that $\mathcal{A}$ can do in the real protocol must be simulated by $\mathcal{S}$ in the ideal execution of the protocol. A point that should be clarified here is that $\mathcal{S}$ does not deliver any message between the parties. It just simulates the messages of the honest parties (if any) to the corrupted party and delivers messages from parties to the ideal functionality. In order to obtain the inputs of the corrupted parties, $\mathcal{S}$ runs an internal copy of the adversary $\mathcal{A}$ with which it communicates using the real protocol.

It is said that a real protocol securely realizes an ideal functionality (i.e., the protocol implementation is secure under the UC framework) if for every adversary in the real protocol $\mathcal{A}$ there exists an ideal adversary $\mathcal{S}$ such that every $\mathcal{Z}$ cannot distinguish an execution of the specific

Functionality $\mathcal{F}_{TI}^{\mathcal{D}}$

$\mathcal{F}_{TI}^{\mathcal{D}}$ runs with parties A and B and is parametrized by the security parameter λ and by an algorithm $\mathcal{D}$.

Alice's Output: Upon receiving a message (sid, A) from A, run $(OP_A, OP_B) \leftarrow \mathcal{D}(1^\lambda)$ if a message (sid, B) has not been received before. Send (sid, A-GETS-OUTPUT) to B and (sid, OP_A) to A.

Bob's Output: Upon receiving a message (sid, B) from B, run $(OP_A, OP_B) \leftarrow \mathcal{D}(1^\lambda)$ if a message (sid, A) has not been received before. Send (sid, B-GETS-OUTPUT) to A and (sid, OP_B) to B.

Figure 1: The Trusted Initializer functionality.

protocol implementation with the parties and $\mathcal{A}$ from an execution of the ideal version with the parties and $\mathcal{S}$. This is stated formally in the following definition from [13]:

Definition 2.1 *A protocol π is said to UC-realize an ideal functionality $\mathcal{F}$ if, for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that, for every environment $\mathcal{Z}$, the following holds:*

$$EXEC_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} IDEAL_{\mathcal{F},\mathcal{S},\mathcal{Z}}$$

Notice that $EXEC_{\pi,\mathcal{A},\mathcal{Z}}$ represents the view of $\mathcal{Z}$ in the real protocol execution with $\mathcal{A}$ and $IDEAL_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ represents the view of $\mathcal{Z}$ in the ideal execution with the simulator $\mathcal{S}$. The probability distribution is taken over the random tapes of the parties.

Obtaining computational indistinguishability between real and ideal executions guarantees that the protocol is secure against PPT adversaries. Even though this is enough for the security requirements of most protocols and applications, it is interesting to achieve security against computationally unbounded adversaries. In this paper we show that the real execution of our protocols is *statistically* indistinguishable from the ideal simulation, thus providing security against attackers that have unlimited computer power.

In this work we consider security against static adversaries, *i.e.* the sets of corrupted and honest parties are fixed before protocol execution and once a party is corrupted or honest it remains so during the whole execution. In addition we use the Truster Initializer as our setup assumption, i.e., it is assumed that the parties involved in the protocol receive data from a trusted initializer before protocol execution. The trusted initializer functionality is described in Figure 1 and is parametrized by an algorithm $\mathcal{D}$ that generates the correlated randomness that is distributed to the two parties, denoted Alice A and Bob B.

3 Vector Inner Product and Matrix Multiplication

In this section, we present a protocol for computing the inner product of two vectors provided by two different parties. Using this protocol as a building block, we then construct a protocol for two-party matrix multiplication. Both protocols were first introduced in [26].

Functionality $\mathcal{F}_{IP}$

$\mathcal{F}_{IP}$ runs with parties A and B and is parametrized by the size q of the field and the length n of the vectors.

Alice's Input: Upon receiving a message (sid, A-INPUT, $\vec{x}$) from A, ignore any subsequent messages from A. If $\vec{x} \notin \mathbb{F}_q^n$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, B-INPUT) message has been received from B, then store $\vec{x}$ and sid, and send the public delayed output (sid, A-INPUT-RECEIVED) to B; else choose $u \overset{\$}{\leftarrow} \mathbb{F}_q$, set $v \leftarrow \langle \vec{x} \cdot \vec{y} \rangle - u$ and send the public delayed outputs (sid, A-GETS-OUTPUT, u) to A and (sid, B-GETS-OUTPUT, v) to B.

Bob's Input: Upon receiving a message (sid, B-INPUT, $\vec{y}$) from B, ignore any subsequent messages from B. If $\vec{y} \notin \mathbb{F}_q^n$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, A-INPUT) message has been received from A, then store $\vec{y}$ and sid, and send the public delayed output (sid, B-INPUT-RECEIVED) to A; else choose $u \overset{\$}{\leftarrow} \mathbb{F}_q$, set $v \leftarrow \langle \vec{x} \cdot \vec{y} \rangle - u$ and send the public delayed outputs (sid, A-GETS-OUTPUT, u) to A and (sid, B-GETS-OUTPUT, v) to B.

Figure 2: The distributed inner product functionality.

3.1 Inner Product

For two vectors $\vec{x} = (x_1, x_2, ..., x_n)$ and $\vec{y} = (y_1, y_2, ..., y_n)$ of elements of $\mathbb{F}_q$, their inner product $\langle \vec{x} \cdot \vec{y} \rangle$ is defined as

$$\langle \vec{x} \cdot \vec{y} \rangle = \sum_{i=1}^{n} x_i y_i.$$

Considering a two-party scenario, we construct a protocol for distributed inner product that outputs two random values in $\mathbb{F}_q$ whose sum is equal to the original inner product of $\vec{x}$ and $\vec{y}$. Each individual party receives only one of the random values. They can recover the inner product value by exchanging their random outputs. Let A and B be the parties involved in the computation, this notion of distributed inner product is defined in Table 1.

	A	B
Inputs	$\vec{x} \in \mathbb{F}_q^n$	$\vec{y} \in \mathbb{F}_q^n$
Outputs	$u \overset{\$}{\leftarrow} \mathbb{F}_q$	$\langle \vec{x} \cdot \vec{y} \rangle - u$

Table 1: Distributed Inner Product

In order to formally argue about the protocol security, we rewrite the previous definition as an inner product ideal functionality $\mathcal{F}_{IP}$ that is described in Figure 2.

The trusted initializer functionality for the inner product protocol is parametrized by the algorithm $\mathcal{D}_{IP}$ that samples $\vec{x_0}, \vec{y_0} \overset{\$}{\leftarrow} \mathbb{F}_q^n$, computes $s_0 \leftarrow \langle \vec{x_0} \cdot \vec{y_0} \rangle$ and outputs $\vec{x_0}$ to A and $(\vec{y_0}, s_0)$ to B. The protocol for computing the distributed inner product of A's input $\vec{x}$ and B's input $\vec{y}$ is described in Figure 3.

Protocol π_{IP}

1. A and B query $\mathcal{F}_{TI}^{\mathcal{D}_{IP}}$ with messages (sid, A) and (sid, B), respectively. A receives $\vec{x_0}$ and B receives $(\vec{y_0}, s_0)$.
2. B computes $\vec{y_1} \leftarrow \vec{y} - \vec{y_0}$ and sends it to A.
3. A aborts if $\vec{y_1} \notin \mathbb{F}_q^n$. A computes $\vec{x_1} \leftarrow \vec{x} + \vec{x_0}$, samples $u \xleftarrow{\$} \mathbb{F}_q$ and computes $u_1 \leftarrow \langle \vec{x} \cdot \vec{y_1} \rangle - u$. Then A sends $(\vec{x_1}, u_1)$ to B and outputs u.
4. B aborts if $\vec{x_1} \notin \mathbb{F}_q^n$ or $u_1 \notin \mathbb{F}_q$. Otherwise, it outputs $v \leftarrow \langle \vec{x_1} \cdot \vec{y_0} \rangle + u_1 - s_0$.

Figure 3: The distributed inner product protocol π_{IP}.

Theorem 3.1 *The distributed inner product protocol π_{IP} described in Figure 3 securely realizes functionality $\mathcal{F}_{IP}$ in the $\mathcal{F}_{TI}^{\mathcal{D}_{IP}}$-hybrid model with unconditional security. I.e., for every static malicious adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$*

$$EXEC_{\pi_{IP},\mathcal{A},\mathcal{Z}} \overset{s}{\approx} IDEAL_{\mathcal{F}_{IP},\mathcal{S},\mathcal{Z}}.$$

Proof: It is straightforward to check the correctness of the protocol.

$$\begin{aligned} v &:= \langle \vec{x_1} \cdot \vec{y_0} \rangle + u_1 - s_0 \\ &= \langle (\vec{x} + \vec{x_0}) \cdot \vec{y_0} \rangle + (\langle \vec{x} \cdot (\vec{y} - \vec{y_0}) \rangle - u) - \langle \vec{x_0} \cdot \vec{y_0} \rangle \\ &= \langle \vec{x} \cdot \vec{y_0} \rangle + \langle \vec{x_0} \cdot \vec{y_0} \rangle + \langle \vec{x} \cdot \vec{y} \rangle - \langle \vec{x} \cdot \vec{y_0} \rangle - u - \langle \vec{x_0} \cdot \vec{y_0} \rangle \\ &= \langle \vec{x} \cdot \vec{y} \rangle - u \end{aligned}$$

Hence the simulation for the cases where both parties are honest or both parties are corrupted is trivial. The cases in which only one of the parties is corrupted will be dealt in the sequence.

Alice Corrupted, Bob Honest. $\mathcal{S}$ runs an internal (embedded) copy of $\mathcal{A}$ denoted $\tilde{\mathcal{A}}$. Observe that Alice is corrupted, so $\mathcal{S}$ has access to Alice's input $\vec{x}$. The interactions of $\tilde{\mathcal{A}}$ with $\mathcal{S}$ are those of Alice in the real protocol with the other parties, $\mathcal{Z}$, the TI, and Bob. Figure 4 describes the behavior of the simulator when the different events happen. In order to distinguish the variables of this simulated execution with $\tilde{\mathcal{A}}$, primes are used.

We make the following observations:

1. Independent of $\mathcal{A}$, $\mathcal{Z}$ can make Bob send no input or an invalid input. In the simulation, $\mathcal{S}$ behavior after sending the message copies this perfectly.
2. Whatever $\tilde{\mathcal{A}}$'s strategy is, it either sends a valid or an invalid message to the simulated Bob. If the message is invalid, both the simulated and the ideal protocol will send INVALID-INPUT to the two parties, which will be forwarded to $\mathcal{Z}$.

Simulator (Alice Corrupted, Bob Honest)

$\mathcal{S}$ reacts in the following way to the varied events that happen during the execution.

Onset of the Simulation: Choose $\vec{x_0}' \xleftarrow{\$} \mathbb{F}_q^n$ and send it to $\tilde{\mathcal{A}}$ when it queries the TI.

Get input $\vec{x}$: $\mathcal{S}$ forwards $\vec{x}$ from $\mathcal{Z}$ to $\tilde{\mathcal{A}}$, i.e. $\vec{x}' \leftarrow \vec{x}$.

Message B-input-received: $\mathcal{S}$ feeds $\vec{y_1}' \xleftarrow{\$} \mathbb{F}_q^n$ to $\tilde{\mathcal{A}}$.

Message $(\vec{x_1}', u_1')$ from $\tilde{\mathcal{A}}$: If the message is invalid, then $\mathcal{S}$ sends something invalid to $\mathcal{F}_{IP}$. Otherwise it sends the message (sid, A-INPUT, $\vec{x}$) to $\mathcal{F}_{IP}$.

Output u': As long as no response from $\mathcal{F}_{IP}$ is received, $\mathcal{S}$ does nothing, even if this means waiting forever. When an INVALID-INPUT message is received, $\mathcal{S}$ forwards it to $\mathcal{Z}$. When an A-GETS-OUTPUT message is received, $\mathcal{S}$ does the following: It lets the functionality deliver the message B-GETS-OUTPUT. $\mathcal{S}$ also intercepts the simulated output u' and verifies if this value is consistent with the input $\vec{x}$, and the simulated messages. If consistent, $\mathcal{S}$ substitutes the simulated output u' for the real output u obtained from $\mathcal{F}_{IP}$ by setting $u_{\mathcal{S}} \leftarrow u$; else it sets $u_{\mathcal{S}} \leftarrow u'$. Then $u_{\mathcal{S}}$ is sent to $\mathcal{Z}$ through Alice's interface.

Figure 4: The simulator for the case where Alice is corrupted and Bob is honest.

3. Even if $\tilde{\mathcal{A}}$ sent a valid message to the simulated Bob, it can still deviate from the protocol by sending a completely different output. Here, deviate means to send an output u' that is not consistent with the input $\vec{x}' = \vec{x}$ and the messages exchanged during the protocol execution. In the case $\tilde{\mathcal{A}}$ did deviate, $\mathcal{S}$ detects this and does nothing, i.e. it forwards the output produced by $\tilde{\mathcal{A}}$ directly to $\mathcal{Z}$.

4. In the case $\tilde{\mathcal{A}}$ did follow the protocol, $\mathcal{S}$ substitutes $\tilde{\mathcal{A}}$'s output u' for the output u obtained from $\mathcal{F}_{IP}$.

If we consider $\tilde{\mathcal{A}}$ (and $\mathcal{A}$) as deterministic algorithms whose probabilism comes from a random tape, it follows that $\tilde{\mathcal{A}}$ behavior is completely determined by these random bits, called s_A, the incoming message $\vec{x_0}'$, the input $\vec{x}$ and the incoming message $\vec{y_1}'$. We already know that the random bits s_A and the input $\vec{x}$ have the same distribution in the real and ideal protocol, because of the way the model is defined.

So in order to show that $\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}\ : EXEC_{\pi_{IP},\mathcal{A},\mathcal{Z}} \overset{s}{\approx} IDEAL_{\mathcal{F}_{IP},\mathcal{S},\mathcal{Z}}$, it suffices to show that the incoming message $\vec{x_0}'$ produced by $\mathcal{S}$ has the same distribution as the incoming message $\vec{x_0}$ produced by the TI in the real protocol. But this is trivial, since both are generated from the same distribution, the uniform distribution on $\mathbb{F}_q^n$.

In addition, we must show that $\vec{y_1}'$ produced by $\mathcal{S}$ and $\vec{y_1}$ sent by Bob have the same distribution. Observe that $\vec{y_1} := \vec{y} - \vec{y_0}$, with $\vec{y_0} \overset{\$}{\leftarrow} \mathbb{F}_q^n$. Since both TI and Bob are honest in the real protocol, it follows that both $\vec{y_1}'$ and $\vec{y_1}$ are generated according to the uniform distribution on $\mathbb{F}_q^n$.

So we conclude that $\tilde{\mathcal{A}}$'s incoming messages in the simulated protocol have a distribution identical to $\mathcal{A}$'s incoming message in the real protocol. It follows therefore that the ideal and real protocol distributions are perfectly indistinguishable from $\mathcal{Z}$ point of view, which completes the proof.

Alice Honest, Bob Corrupted. The proof of this case is very much along the same lines as the previous case:$\mathcal{S}$ runs an internal (embedded) copy of $\mathcal{A}$ called $\tilde{\mathcal{A}}$. Observe that Bob is corrupted, so $\mathcal{S}$ has access to Bob's input $\vec{y}$. The interactions of $\tilde{\mathcal{A}}$ with $\mathcal{S}$ are those of Bob in the real protocol with the other parties, $\mathcal{Z}$, the TI, and Alice. Figure 5 describes the behavior of the simulator when the different events happen.

The proof of indistinguishability is almost identical to the previous case and is omitted. ▮

3.2 Matrix Multiplication

Matrix multiplication basically consists in computing inner products of the rows and columns of the given matrices. Hence, it is fairly straightforward to realize matrix multiplication from a vector inner product functionality. Analogously to the inner product operation described in Section 3.1, a distributed matrix multiplication operation takes as input two matrices L and M, outputting two random matrices R and C whose sum is equal to the product of L and M. This operation is defined in Table 2.

In order to formally argue about protocol security, we rewrite the previous definition as an ideal functionality $\mathcal{F}_{MM}$ in Figure 6. Note that it is a straightforward adaption of $\mathcal{F}_{IP}$.

A distributed matrix multiplication protocol is described in Figure 7. It assumes access to the vector inner product functionality $\mathcal{F}_{IP}$, and takes matrix L as input from A and matrix M as input from B. It can be instantiated based on Protocol π_{IP}. In this case, the algorithm specifying the outputs of the TI, $\mathcal{D}_{MM}$, will simply be the one generating the correlated randomness

Simulator (Alice Honest, Bob Corrupted)

$\mathcal{S}$ reacts in the following way to the varied events that happen during the execution.

Onset of the Simulation: $\mathcal{S}$ sets $\vec{y_0}' \xleftarrow{\$} \mathbb{F}_q^n$, $s_0' \xleftarrow{\$} \mathbb{F}_q$ and feeds $(\vec{y_0}', s_0')$ to $\tilde{\mathcal{A}}$.

Get input $\vec{y}$: $\mathcal{S}$ forwards $\vec{y}$ from $\mathcal{Z}$ to $\tilde{\mathcal{A}}$, i.e. $\vec{y}' \leftarrow \vec{y}$.

Message $\vec{y_1}'$ from $\tilde{\mathcal{A}}$: If $\tilde{\mathcal{A}}$ sends something invalid, then $\mathcal{S}$ sends something invalid to $\mathcal{F}_{IP}$. $\mathcal{S}$ sends the message (*sid*, B-INPUT, $\vec{y}$) to $\mathcal{F}_{IP}$. $\mathcal{S}$ lets the functionality deliver the message B-INPUT-RECEIVED.

Message (*sid*, B-gets-output, v): $\mathcal{S}$ sets $(\vec{x_1}' \xleftarrow{\$} \mathbb{F}_q^n, u_1' \xleftarrow{\$} \mathbb{F}_q^n$ and feeds $(\vec{x_1}', u_1')$ to $\tilde{\mathcal{A}}$; $\mathcal{S}$ lets the functionality deliver the message A-GETS-OUTPUT. $\mathcal{S}$ intercepts the simulated output $v' = \langle \vec{x_1}' \cdot \vec{y_0}' \rangle + u_1' - s_0'$ and verifies if this value is consistent with the input $\vec{y}$, and the simulated messages. If consistent, $\mathcal{S}$ substitutes the simulated output v' with the real output v obtained from $\mathcal{F}_{IP}$ by setting $v_{\mathcal{S}} \leftarrow v$; else it sets $v_{\mathcal{S}} \leftarrow v'$. As long as no v' from $\tilde{\mathcal{A}}$ is received, $\mathcal{S}$ does not forward the message B-GETS-OUTPUT, even if this means waiting forever. Then $v_{\mathcal{S}}$ is sent to $\mathcal{Z}$ through Bob's interface.

Figure 5: The simulator for the case where Alice is honest and Bob is corrupted.

	A	B
Inputs	$L \in \mathbb{F}_q^{i \times j}$	$M \in \mathbb{F}_q^{j \times k}$
Outputs	$R \xleftarrow{\$} \mathbb{F}_q^{i \times k}$	$C \leftarrow LM - R$

Table 2: Distributed Matrix Multiplication

Functionality $\mathcal{F}_{MM}$

$\mathcal{F}_{MM}$ runs with parties A and B and is parametrized by the size q of the field and the dimensions of the matrices.

Alice's Input: Upon receiving a message (*sid*, A-INPUT, L) from A, ignore any subsequent messages from A. If $L \notin \mathbb{F}_q^{i \times j}$, then send the message (*sid*, INVALID-INPUT) to A and B and stop. If no (*sid*, B-INPUT) message has been received from B, then store L and *sid*, and send the public delayed output (*sid*, A-INPUT-RECEIVED) to B; else choose $R \xleftarrow{\$} \mathbb{F}_q^{i \times k}$, set $C \leftarrow LM - R$ and send the public delayed outputs (*sid*, A-GETS-OUTPUT, R) to A and (*sid*, B-GETS-OUTPUT, C) to B.

Bob's Input: Upon receiving a message (*sid*, B-INPUT, M) from B, ignore any subsequent messages from B. If $M \notin \mathbb{F}_q^{j \times k}$, then send the message (*sid*, INVALID-INPUT) to A and B and stop. If no (*sid*, A-INPUT) message has been received from A, then store M and *sid*, and send the public delayed output (*sid*, B-INPUT-RECEIVED) to A; else choose $R \xleftarrow{\$} \mathbb{F}_q^{i \times k}$, set $C \leftarrow LM - R$ and send the public delayed outputs (*sid*, A-GETS-OUTPUT, R) to A and (*sid*, B-GETS-OUTPUT, C) to B.

Figure 6: The distributed matrix multiplication functionality.

necessary to the $i \cdot k$ independent executions of π_{IP}. Once the correlated randomness are pre-distributed by the TI, the instances of the underlying inner product protocol can be executed in parallel, yielding an efficient protocol with low round complexity. This matrix multiplication protocol will be used as a building block for the other constructions presented in this paper.

Theorem 3.2 *The distributed matrix multiplication protocol π_{MM} described in Figure 7 securely realizes functionality $\mathcal{F}_{MM}$ in the $\mathcal{F}_{IP}$-hybrid model with unconditional security. I.e., for every static malicious adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$*

$$EXEC_{\pi_{MM},\mathcal{A},\mathcal{Z}} \overset{s}{\approx} IDEAL_{\mathcal{F}_{MM},\mathcal{S},\mathcal{Z}}.$$

Proof: The correctness of the protocol follows trivially from the correctness of the distributed inner product functionality. Notice that no cryptographic operations are performed in the previous protocol, except for calls to the distributed inner product functionality. Hence, the security of this protocol follows very straightforwardly from the UC composition theorem [13]. Intuitively, the simulator acts as $\mathcal{F}_{IP}$ when interacting with corrupted parties, having access

Protocol π_{MM}

1. For all $1 \leq n \leq i$, $1 \leq p \leq k$, A and B do the following:
   - A selects the n-th line of L, denoted as $\vec{\ell_n}$, and sends (sid, A-INPUT, $\vec{\ell_n}$) to an instance of $\mathcal{F}_{IP}$ with n and p encoded in sid.
   - B selects the p-th column of M, denoted as $\vec{m_p}$, and sends (sid, B-INPUT, $\vec{m_p}$) to an instance of $\mathcal{F}_{IP}$ with n and p encoded in sid.
2. In order to obtain the final matrix product, A and B do the following:
   - Upon receiving all the messages A-GETS-OUTPUT, A constructs matrix R by setting $r_{n,p} \leftarrow u_{n,p}$, where for all $1 \leq n \leq i$, $1 \leq p \leq k$, $u_{n,p}$ is the output received from the instance of $\mathcal{F}_{IP}$ corresponding to this n and p.
   - Upon receiving all the messages B-GETS-OUTPUT, B constructs matrix C by setting $c_{n,p} \leftarrow v_{n,p}$, where for all $1 \leq n \leq i$, $1 \leq p \leq k$, $v_{n,p}$ is the output received from the instance of $\mathcal{F}_{IP}$ corresponding to this n and p.

Figure 7: The distributed matrix multiplication protocol π_{MM}.

to the inputs sent by corrupt parties to $\mathcal{F}_{IP}$. The simulator extracts their input matrices by aggregating the row (or column) vectors sent to $\mathcal{F}_{IP}$ during execution. ▌

4 Solving Linear Equations

We now show how to use the previously constructed matrix multiplication protocol in order to obtain a new protocol for securely solving linear equations. The functionality is described in Figure 8.

The solution is based on Du's approach [30]. Note that the solution $\vec{z}$ to the linear equation $(L+M)\vec{z} = \vec{x} + \vec{y}$ is equal to the solution $\vec{z}$ in $P(L+M)QQ^{-1}\vec{z} = P(\vec{x}+\vec{y})$, in which P and Q are random, invertible matrices over $\mathbb{F}_q$ only known by Bob. In the protocol, we let Alice solve the blinded equation $P(L+M)Q\vec{t} = P(\vec{x}+\vec{y})$, so $t = Q^{-1}\vec{z}$. In other words, the solution that Alice gets to see is the final solution $\vec{z}$, blinded with a random invertible matrix Q^{-1}. To allow Alice to compute $P(L+M)Q$ and $P(\vec{x}+\vec{y})$ without her learning M or $\vec{y}$, we use the distributed matrix multiplication as a subprotocol. Though the protocol notation below seems to suggest otherwise, it should be pointed out that when the subprotocols are instantiated, the initialization phase for the main protocol and the subprotocols takes place at the same time. The TI functionality is in this case similar to the previous one, but in addition to the data used by the matrix multiplication protocols, it also pre-distributes the other data needed by the protocol described in Figure 9.

Theorem 4.1 *The linear equation solver protocol π_{LES} described in Figure 9 securely realizes functionality $\mathcal{F}_{LES}$ in the $\mathcal{F}_{MM}, \mathcal{F}_{TI}^{\mathcal{D}_{LES}}$-hybrid model with unconditional security. I.e., for every static malicious adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$*

$$EXEC_{\pi_{LES},\mathcal{A},\mathcal{Z}} \overset{s}{\approx} IDEAL_{\mathcal{F}_{LES},\mathcal{S},\mathcal{Z}}.$$

Functionality $\mathcal{F}_{LES}$

$\mathcal{F}_{LES}$ runs with parties A and B and is parametrized by the size q of the field and n.

Alice's Input: Upon receiving a message (sid, A-INPUT, L, $\vec{x}$) from A, ignore any subsequent messages from A. If $L \notin SL(\mathbb{F}_q, n)$ or $\vec{x} \notin \mathbb{F}_q^n$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, B-INPUT) message has been received from B, then store L, $\vec{x}$ and sid, and send the public delayed output (sid, A-INPUT-RECEIVED) to B; else find $\vec{z}$ such that $(L + M)\vec{z} = \vec{x} + \vec{y}$ and send the public delayed output (sid, B-GETS-OUTPUT, $\vec{z}$) to B.

Bob's Input: Upon receiving a message (sid, B-INPUT, M, $\vec{y}$) from B, ignore any subsequent messages from B. If $M \notin SL(\mathbb{F}_q, n)$ or $\vec{y} \notin \mathbb{F}_q^n$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, A-INPUT) message has been received from A, then store M, $\vec{y}$ and sid, and send the public delayed output (sid, B-INPUT-RECEIVED) to A; else find $\vec{z}$ such that $(L + M)\vec{z} = \vec{x} + \vec{y}$ and send the public delayed output (sid, B-GETS-OUTPUT, $\vec{z}$) to B.

Figure 8: The linear equation solver functionality.

Protocol π_{LES}

1. For $Q \xleftarrow{\$} SL(\mathbb{F}_q, n)$ and $R, U \xleftarrow{\$} \mathbb{F}_q^{n\times n}$, A gets $(R, V = RQ + U)$ from the TI and B gets (Q, U).

2. B generates $P \xleftarrow{\$} SL(\mathbb{F}_q, n)$. B and A interact using $\mathcal{F}_{MM}$ with inputs P and L, respectively, and receive the outputs $PL - \tilde{R}$ and $\tilde{R}$, respectively. In parallel they run another instance of $\mathcal{F}_{MM}$ with inputs P and $\vec{x}$, respectively, and receive the outputs $P\vec{x} - \vec{s}$ and $\vec{s}$, respectively.

3. A send the matrix $R - \tilde{R}$ to B.

4. B computes $W \leftarrow (PL - R)Q + PMQ - U$ and $\vec{c} \leftarrow (P\vec{x} - \vec{s}) + P\vec{y}$, and sends $(W, \vec{c})$ to A.

5. A check if the message is valid, aborting if it is not. Otherwise, she finds $\vec{t}$ such that $(W + V)\vec{t} = \vec{c} + \vec{s}$ and sends it to B.

6. Bob computes $\vec{z} = Q\vec{t}$.

Figure 9: The linear equation solver protocol π_{LES}.

Simulator (Alice Corrupted, Bob Honest)

$\mathcal{S}$ reacts in the following way to the varied events that happen during the execution.

Onset of the Simulation: Choose the pre-distributed data following the correct procedures of the trusted initializer functionality and sends Alice's pre-distributed data to $\tilde{\mathcal{A}}$.

Get input $(L, \vec{x})$**:** $\mathcal{S}$ forwards $(L, \vec{x})$ from $\mathcal{Z}$ to $\tilde{\mathcal{A}}$, i.e. $L' \leftarrow L$ and $\vec{x}' \leftarrow \vec{x}$. $\mathcal{S}$ also chooses $\vec{y}' \overset{\$}{\leftarrow} \mathbb{F}_q^n$ and $M', P' \overset{\$}{\leftarrow} SL(\mathbb{F}_q, n)$.

Message B**-input-received:** $\mathcal{S}$ computes W' and $\vec{c}'$, and sends $(W', \vec{c}')$ to $\tilde{\mathcal{A}}$.

Message $\vec{t}'$ **from** $\tilde{\mathcal{A}}$**:** If the message is invalid, then $\mathcal{S}$ sends something invalid to $\mathcal{F}_{LES}$. Otherwise it sends the message (sid, A-INPUT, L, $\vec{x}$) to $\mathcal{F}_{LES}$.

Message B**-gets-output:** $\mathcal{S}$ lets $\mathcal{F}_{LES}$ deliver the message.

Figure 10: The simulator for the case where Alice is corrupted and Bob is honest.

Proof: The correctness of the protocol is trivially verified: $W + V = (PL - R)Q + PMQ - U + RQ + U = P(L + M)Q$ and $\vec{c} + \vec{s} = (P\vec{x} - \vec{s}) + P\vec{y} + \vec{s} = P(\vec{x} + \vec{y})$, so Alice solves the equation $P(L + M)Q\vec{t} = P(\vec{x} + \vec{y})$. In order to find the solution to $(X + Y)\vec{z} = \vec{x} + \vec{y}$, Bob only has to compute $\vec{z} = Q\vec{t}$. The simulation for the cases where both parties are honest or both parties are corrupted is trivial. The cases in which only one of the parties is corrupted will be dealt in the sequence.

Alice Corrupted, Bob Honest. $\mathcal{S}$ runs an internal (embedded) copy of $\mathcal{A}$ called $\tilde{\mathcal{A}}$. Observe that Alice is corrupted, so $\mathcal{S}$ has access to Alice's inputs $\vec{x}$ and L. The interactions of $\tilde{\mathcal{A}}$ with $\mathcal{S}$ are those of Alice in the real protocol with the other parties, $\mathcal{Z}$, the TI, and Bob. Figure 10 describes the behavior of the simulator when the different events happen.

We make the following observations:

1. Independent of $\mathcal{A}$, $\mathcal{Z}$ can make Bob send no input or an invalid input. In the simulation, $\mathcal{S}$ behavior after sending the message copies this perfectly.
2. Whatever $\tilde{\mathcal{A}}$'s strategy is, it either sends a valid or an invalid message as being A's one. If the message is invalid, both the simulated and the ideal protocol will send INVALID-INPUT to the two parties, which will be forwarded to $\mathcal{Z}$.

If we consider $\tilde{\mathcal{A}}$ (and $\mathcal{A}$) as deterministic algorithms whose probabilism comes from a random tape, it follows that $\tilde{\mathcal{A}}$ behavior is completely determined by these random bits, called s_A, the pre-distributed data, the inputs L and $\vec{x}$, and the incoming message $(W', \vec{c}')$. We already know that the random bits s_A and the inputs L and $\vec{x}$ have the same distribution in the real and ideal protocol, because of the way the model is defined.

Simulator (Alice Honest, Bob Corrupted)

$\mathcal{S}$ reacts in the following way to the varied events that happen during the execution.

Onset of the Simulation: Choose the pre-distributed data following the correct procedures of the trusted initializer Functionality and sends Bob's pre-distributed data to $\tilde{\mathcal{A}}$.

Get input $(M, \vec{y})$**:** $\mathcal{S}$ forwards $(M, \vec{y})$ from $\mathcal{Z}$ to $\tilde{\mathcal{A}}$, i.e. $M' \leftarrow M$ and $\vec{y}' \leftarrow \vec{y}$. $\mathcal{S}$ also chooses $\vec{x}' \xleftarrow{\$} \mathbb{F}_q^n$ and $L' \xleftarrow{\$} SL(\mathbb{F}_q, n)$.

Message $(M', \vec{c}')$ **from** $\tilde{\mathcal{A}}$**:** If the message is invalid, then $\mathcal{S}$ sends something invalid to $\mathcal{F}_{LES}$. Otherwise it sends the message (sid, A-INPUT, M, $\vec{y}$) to $\mathcal{F}_{LES}$. $\mathcal{S}$ lets the functionality deliver the message B-INPUT-RECEIVED.

Message (*sid*, B-gets-output, $\vec{z}$): $\mathcal{S}$ computes t' and sends it to $\tilde{\mathcal{A}}$. $\mathcal{S}$ intercepts the simulated output $\vec{z}'$ and verifies if this value is consistent with the values used in this simulated execution (note that $\mathcal{S}$ knows all these values as it plays the role of the trusted initializer). If consistent, $\mathcal{S}$ substitutes the simulated output $\vec{z}'$ with the real output $\vec{z}$ obtained from $\mathcal{F}_{LES}$ by setting $\vec{z}_{\mathcal{S}} = \vec{z}$; else it sets $\vec{z}_{\mathcal{S}} = \vec{z}'$. As long as no $\vec{z}'$ from $\tilde{\mathcal{A}}$ is received, $\mathcal{S}$ does not forward the message B-GETS-OUTPUT, even if this means waiting forever. Then $\vec{z}_{\mathcal{S}}$ is sent to $\mathcal{Z}$ through Bob's interface.

Figure 11: The simulator for the case where Alice is honest and Bob is corrupted.

So in order to show that $\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}\ : EXEC_{\pi_{LES},\mathcal{A},\mathcal{Z}} \overset{s}{\approx} IDEAL_{\mathcal{F}_{LES},\mathcal{S},\mathcal{Z}}$, it suffices to show that the pre-distributed data produced by $\mathcal{S}$ has the same distribution as the pre-distributed data produced by the TI in the real protocol. But this is trivial, since $\mathcal{S}$ generate these data using the same distribution that the TI uses in the real protocol.

In addition, we must show that $(W', \vec{c}')$ produced by $\mathcal{S}$ and $(W, \vec{c})$ sent by Bob have the same distribution. Observe that $M := (PL - R)Q + PMQ - U$ with $U \xleftarrow{\$} \mathbb{F}_q^{n\times n}$ and $\vec{c} := (P\vec{x} - \vec{s}) + P\vec{y}$ with $P \xleftarrow{\$} SL(\mathbb{F}_q, n)$. Since both TI and Bob are honest in the real protocol, it follows that both $(W', \vec{c}')$ and $(W, \vec{c})$ are generated according to the uniform distribution.

So we conclude that $\tilde{\mathcal{A}}$'s incoming messages in the simulated protocol have a distribution identical to $\mathcal{A}$'s incoming message in the real protocol. It follows therefore that the ideal and real protocol distributions are perfectly indistinguishable from $\mathcal{Z}$ point of view, which completes the proof for this case.

Alice Honest, Bob Corrupted The proof of this case is very much along the same lines as the previous case: $\mathcal{S}$ runs an internal (embedded) copy of $\mathcal{A}$ called $\tilde{\mathcal{A}}$. Observe that Bob is corrupted, so $\mathcal{S}$ has access to Bob's input $\vec{y}$. The interactions of $\tilde{\mathcal{A}}$ with $\mathcal{S}$ are those of Bob in the real protocol with the other parties, $\mathcal{Z}$, the TI, and Alice. We explain how $\mathcal{S}$ acts in Figure 11.

The proof of indistinguishability is almost identical to the previous case and is omitted. ∎

Functionality $\mathcal{F}_{DET}$

$\mathcal{F}_{DET}$ runs with parties A and B and is parametrized by the size q of the field and n.

Alice's Input: Upon receiving a message (sid, A-INPUT, L) from A, ignore any subsequent messages from A. If $L \notin \mathbb{F}_q^{n\times n}$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, B-INPUT) message has been received from B, then store L and sid, and send the public delayed output (sid, A-INPUT-RECEIVED) to B; else compute $z \leftarrow \det(L+M)$ and send the public delayed output (sid, B-GETS-OUTPUT, z) to B.

Bob's Input: Upon receiving a message (sid, B-INPUT, M) from B, ignore any subsequent messages from B. If $M \notin \mathbb{F}_q^{n\times n}$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, A-INPUT) message has been received from A, then store M and sid, and send the public delayed output (sid, B-INPUT-RECEIVED) to A; else compute $z \leftarrow \det(L+M)$ and send the public delayed output (sid, B-GETS-OUTPUT, z) to B.

Figure 12: The distributed determinant functionality.

5 Determinant

In this section we present a two-party protocol for computing the determinant of a sum of two matrices. This protocol relies on access to a distributed matrix multiplication functionality $\mathcal{F}_{MM}$, and can be instantiated with the protocol described in Section 3.2.

We consider a distributed two-party determinant operation where players A and B input matrices L and M, respectively. The operation outputs the determinant of $L+M$ to B and $\perp$ to A (*i.e.* A does not learn anything). This operation is illustrated in Table 3. In order to formally argue about protocol security, the definition is rewritten as an ideal functionality $\mathcal{F}_{DET}$ in Figure 12.

	A	B
Inputs	$L \in \mathbb{F}_q^{n\times n}$	$M \in \mathbb{F}_q^{n\times n}$
Outputs	$\perp$	$\det(L+M)$

Table 3: Distributed Determinant

We present in Figure 13 a protocol π_{DET} that implements such functionality. It is essentially a simplified version of protocol π_{LES} for solving linear equations (see Section 4), with trivial modifications in the last steps (i.e., the blinded output forward from A to B and the computation of B's output). It also assumes access to a distributed matrix multiplication functionality $\mathcal{F}_{MM}$ and to a trusted initializer functionality $\mathcal{F}_{TI}^{\mathcal{D}_{DET}}$, which also for $Q \overset{\$}{\leftarrow} SL(\mathbb{F}_q, n)$ and $U, R \overset{\$}{\leftarrow} \mathbb{F}_q^{n\times n}$ outputs $(R, V = RQ+U)$ to A and (Q, U) to B. Also in this case when $\mathcal{F}_{MM}$ is instantiated using π_{MM}, the pre-distribution phase performed by the TI occurs at the same time for the data needed by π_{DET} and by π_{MM}.

The intuition for the security of the protocol is that A cannot extract M from W because it is "hidden" by the multiplication between matrices P and Q which are unknown to A. In

Protocol π_{DET}

1. For $Q \xleftarrow{\$} SL(\mathbb{F}_q, n)$ and $U, R \xleftarrow{\$} \mathbb{F}_q^{n\times n}$, A gets $(R, V = RQ + U)$ from the TI and B gets (Q, U).
2. B generates $P \xleftarrow{\$} SL(\mathbb{F}_q, n)$. B and A interact using $\mathcal{F}_{MM}$ with inputs P and L, respectively, and receive the outputs $PL - \tilde{R}$ and $\tilde{R}$, respectively.
3. A send the matrix $R - \tilde{R}$ to B.
4. B computes $W \leftarrow (PL - R)Q + PMQ - U$ and sends it to A.
5. A check if $W \notin \mathbb{F}_q^{n\times n}$, aborting if it is not. Otherwise, she computes $t = \det(W + V)$ and sends it to B.
6. Bob computes $z = \frac{t}{\det(P)\det(Q)}$ and outputs it.

Figure 13: The protocol π_{DET} for computing the determinant.

addition, the terms $(PL - R)$ and U are unknown to A as well. Since the resulting determinant is a number in $\mathbb{F}_q$ and, in general, $\det(M_1 + M_2) \neq \det(M_1) + \det(M_2)$, B cannot extract L from his output, neither from t.

Theorem 5.1 *The protocol π_{DET} for computing the determinant described in Figure 13 securely realizes functionality $\mathcal{F}_{DET}$ in the $\mathcal{F}_{MM}, \mathcal{F}_{TI}^{\mathcal{D}_{DET}}$-hybrid model with unconditional security. I.e., for every static malicious adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$*

$$EXEC_{\pi_{DET},\mathcal{A},\mathcal{Z}} \overset{s}{\approx} IDEAL_{\mathcal{F}_{DET},\mathcal{S},\mathcal{Z}}.$$

Proof: The correctness of the protocol can be verified as follows. Note that

$$W + V = (PL - R)Q + PMQ - U + RQ + U = P(L + M)Q.$$

It is known that for any square matrices M_1 and M_2, the following relation holds $\det(M_1M_2) = \det(M_1)\det(M_2)$. Moreover, $\det(P) \neq 0$ and $\det(Q) \neq 0$ since P and Q are non-singular matrices. Hence the following relation holds:

$$z = \frac{t}{\det(P)\det(Q)} = \frac{\det(P(L + M)Q)}{\det(P)\det(Q)} = \det(L + M).$$

The simulation for the case where both parties are honest is trivial. The same holds for the case where both parties are corrupted. In the remaining cases, the description of $\mathcal{S}$'s actions are essentially a simplified version of that in Section 4 (with the trivial adaptions needed) and is thus omitted. ▌

Functionality $\mathcal{F}_{Eigenvalue}$

$\mathcal{F}_{Eigenvalue}$ runs with parties A and B and is parametrized by the size q of the field and n.

Alice's Input: Upon receiving a message (sid, A-INPUT, L) from A, ignore any subsequent messages from A. If $L \notin \mathbb{F}_q^{n\times n}$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, B-INPUT) message has been received from B, then store L and sid, and send the public delayed output (sid, A-INPUT-RECEIVED) to B; else determine the solutions of the equation $\det(L + M - \lambda I) = 0$, form a vector $\vec{\lambda}$ with them and send the public delayed output (sid, A-GETS-OUTPUT, $\vec{\lambda}$) to A.

Bob's Input: Upon receiving a message (sid, B-INPUT, M) from B, ignore any subsequent messages from B. If $M \notin \mathbb{F}_q^{n\times n}$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, A-INPUT) message has been received from A, then store M and sid, and send the public delayed output (sid, B-INPUT-RECEIVED) to A; else determine the solutions of the equation $\det(L + M - \lambda I) = 0$, form a vector $\vec{\lambda}$ with them and send the public delayed output (sid, A-GETS-OUTPUT, $\vec{\lambda}$) to A.

Figure 14: The distributed eigenvalues functionality.

6 Eigenvalue

Using the same protocol structure as in the previous sections, it is possible to design a protocol that computes the eigenvalues corresponding to the sum of two matrices in a private way (i.e., without revealing any additional information about the matrices). Note that such protocols will obviously have the same limitations (regarding the types of matrices/output for which the computation can be done efficiently) as the traditional algorithms for finding the eigenvalues. In this case Alice is the one getting the output, since there is no blinded output involved (but the parties can be reversed). The formal functionality is presented in Figure 14 and the protocol realizing it in Figure 15.

Theorem 6.1 *The protocol $\pi_{Eigenvalue}$ for computing the eigenvalues described in Figure 15 securely realizes functionality $\mathcal{F}_{Eigenvalue}$ in the $\mathcal{F}_{MM}, \mathcal{F}_{TI}^{\mathcal{D}_{Eigenvalue}}$-hybrid model with unconditional security. I.e., for every static malicious adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$*

$$EXEC_{\pi_{Eigenvalue},\mathcal{A},\mathcal{Z}} \overset{s}{\approx} IDEAL_{\mathcal{F}_{Eigenvalue},\mathcal{S},\mathcal{Z}}.$$

Proof: The correctness of the protocol can be verified as follows. Note that

$$W + V = (PL - R)Q + PTQ - U + RQ + U = P(L + M - \lambda I)Q.$$

Since P and Q are non-singular matrices, $\det(P) \neq 0$ and $\det(Q) \neq 0$. Therefore

$$\begin{aligned} \det(W + V) = 0 \quad &\Leftrightarrow \quad \det(P(L + M - \lambda I)Q) = 0 \\ &\Leftrightarrow \quad \det(P)\det(L + M - \lambda I)\det(Q) = 0 \\ &\Leftrightarrow \quad \det(L + M - \lambda I) = 0. \end{aligned}$$

Protocol $\pi_{Eigenvalue}$

1. For $Q \xleftarrow{\$} SL(\mathbb{F}_q, n)$ and $U, R \xleftarrow{\$} \mathbb{F}_q^{n\times n}$, A gets $(R, V = RQ + U)$ from the TI and B gets (Q, U).
2. B generates $P \xleftarrow{\$} SL(\mathbb{F}_q, n)$. B and A interact using $\mathcal{F}_{MM}$ with inputs P and L, respectively, and receive the outputs $PL - \tilde{R}$ and $\tilde{R}$, respectively.
3. A send the matrix $R - \tilde{R}$ to B.
4. Considering λ as a variable, B computes $T \leftarrow M - \lambda I$ and $W \leftarrow (PL - R)Q + PTQ - U$, and sends W to A.
5. A check if $W \notin \mathbb{F}_q^{n\times n}$, aborting if it is not. Otherwise, she computes the solutions of the equation $\det(W + V) = 0$ and outputs a vector containing them.

Figure 15: The protocol $\pi_{Eigenvalue}$ for computing the eigenvalues.

The description of the simulator is similar to the ones in the previous sections and is thus omitted. ▮

7 Eigenvector

It is also possible to design a protocol that enables Bob to obtain an eigenvector corresponding to a specific eigenvalue. The functionality is described in Figure 16 and the protocol realizing it in Figure 17.

Theorem 7.1 *The protocol $\pi_{Eigenvector}$ for computing an eigenvector described in Figure 17 securely realizes functionality $\mathcal{F}_{Eigenvector}$ in the $\mathcal{F}_{MM}, \mathcal{F}_{TI}^{\mathcal{D}_{Eigenvector}}$-hybrid model with unconditional security. I.e., for every static malicious adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$*

$$EXEC_{\pi_{Eigenvector}, \mathcal{A}, \mathcal{Z}} \overset{s}{\approx} IDEAL_{\mathcal{F}_{Eigenvector}, \mathcal{S}, \mathcal{Z}}.$$

Proof: The correctness of the protocol can be verified as follows. We have that

$$W + V = (PL - R)Q + PTQ - U + RQ + U = P(L + M - \lambda_i I)Q.$$

Therefore A calculates $P(L + M - \lambda_i I)Q\vec{t} = P\vec{0} \Rightarrow (L + M - \lambda_i I)Q\vec{t}$ and B can find the solution of the equation $(L + M - \lambda_i I)\vec{z} = 0$, by computing $\vec{z} = Q\vec{t}$.

The description of the simulator is similar to the ones presented before. ▮

8 Complexity and Efficiency

In order to assess the concrete efficiency and practicality of our protocols, we analyse the concrete performance of a prototype implementation of our protocol suite. Our protocols are implemented

Functionality $\mathcal{F}_{Eigenvector}$

$\mathcal{F}_{Eigenvector}$ runs with parties A and B and is parametrized by the size q of the field and n.

Alice's Input: Upon receiving a message (sid, A-INPUT, L) from A, ignore any subsequent messages from A. If $L \notin \mathbb{F}_q^{n\times n}$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, B-INPUT) message has been received from B, then store L and sid, and send the public delayed output (sid, A-INPUT-RECEIVED) to B; else check if λ_i is an eigenvalue of $L+M$ and send the message (sid, INVALID-INPUT) to A and B and stop if it is not. If it is an eigenvalue, compute $\vec{z}$ that is an eigenvector corresponding to the matrix $L+M$ and eigenvalue λ_i, and send the public delayed output (sid, A-GETS-OUTPUT, $\vec{z}$) to B.

Bob's Input: Upon receiving a message (sid, B-INPUT, M, λ_i) from B, ignore any subsequent messages from B. If $M \notin \mathbb{F}_q^{n\times n}$, then send the message (sid, INVALID-INPUT) to A and B and stop. If no (sid, A-INPUT) message has been received from A, then store M, λ_i and sid, and send the public delayed output (sid, B-INPUT-RECEIVED) to A; else check if λ_i is an eigenvalue of $L+M$ and send the message (sid, INVALID-INPUT) to A and B and stop if it is not. If it is an eigenvalue, compute $\vec{z}$ that is an eigenvector corresponding to the matrix $L+M$ and eigenvalue λ_i, and send the public delayed output (sid, A-GETS-OUTPUT, $\vec{z}$) to B.

Figure 16: The distributed eigenvector functionality.

Protocol $\pi_{Eigenvector}$

1. For $Q \xleftarrow{\$} SL(\mathbb{F}_q, n)$ and $U, R \xleftarrow{\$} \mathbb{F}_q^{n\times n}$, A gets $(R, V = RQ + U)$ from the TI and B gets (Q, U).
2. B generates $P \xleftarrow{\$} SL(\mathbb{F}_q, n)$. B and A interact using $\mathcal{F}_{MM}$ with inputs P and L, respectively, and receive the outputs $PL - \tilde{R}$ and $\tilde{R}$, respectively.
3. A send the matrix $R - \tilde{R}$ to B.
4. Bob computes $T \leftarrow M - \lambda_i I$ and $W \leftarrow (PL - R)Q + PTQ - U$, and sends W to A.
5. A check if $W \notin \mathbb{F}_q^{n\times n}$, aborting if it is not. Otherwise, she computes $\vec{t}$ such that $(W+V)\vec{t} = 0$ and sends it to B.
6. Bob computes $\vec{z} = Q\vec{t}$ and outputs it.

Figure 17: The protocol $\pi_{Eigenvector}$ for computing an eigenvector.

over $\mathbb{Z}_p$ with $p = 65521$ and the execution time of each protocol is evaluated for different input sizes. In all of our experiments, we execute each protocol 50 times for each input size and determine the average execution time. The execution times of each protocol are described in table 4.

The protocols are implemented in C++ using the libraries NTL 6.1 for algebraic operations over $\mathbb{Z}_p$ and Boost 1.55 for asynchronous network communication. The performance of the implementations was evaluated in two computers with an Intel(R) Xeon(R) E7-2850 processor at 2.00GHz and 8 GB RAM running Linux Kernel 3.2.0-88 connected through a Gigabit Ethernet network. Even though the CPUs used in the evaluation hosts have four cores, the implementation uses only one thread.

n	π_{MM} (s)	π_{LES} (s)	π_{DET} (s)	$\pi_{Eigenvalue}$ (s)	$\pi_{Eigenvector}$ (s)
50	3.126	6.273	3.145	3.147	3.144
60	4.937	9.910	4.969	4.970	4.967
70	6.120	12.294	6.168	6.170	6.168
80	8.987	18.054	9.059	9.062	9.058
90	12.460	25.034	12.563	12.567	12.559
100	13.239	26.643	13.385	13.389	13.373

Table 4: Execution times (in seconds) of protocols π_{MM}, π_{LES}, π_{DET}, $\pi_{Eigenvalue}$, $\pi_{Eigenvector}$ taking as input a $n \times n$ square matrix.

As remarked before, our implementations are not optimized, thus not showcasing the full potential of our protocols. The main issue in these implementations is the overhead of performing I/O with the NTL library, which converts numbers into an ASCII representation that greatly increases concrete communication complexity. Hence, we devote special attention to the distributed inner product protocol π_{IP}, which is the core of our constructions for other distributed linear algebra operations. We analyse the concrete performance of an optimized implementation of π_{IP} using only native C types and operators for the necessary algebraic operations. In this scenario, we consider as input vectors with different numbers of 16 bit elements (represented by the uint16_t type). The concrete performance of this optimized implementation with different input sizes is analysed and compared to the performance of a trivial protocol that computes the same functionality without guaranteeing any security. The computational overhead of optimized π_{IP} over the trivial protocol is calculated as the ratio between the execution times of π_{IP} and the trivial protocol. The resulting data can be seen in Table 5.

From the data presented in Table 5 it is clear that our approaches guarantee security in distributed linear algebra operations incurring only a very low overhead for small datasets. In the specific case of π_{IP}, as the datasets grow, the overhead also grows but it still requires less than double the execution time of a trivial implementation of the same functionality. Moreover, notice that the optimized implementation is significantly faster than the NTL based implementation. These data provide evidence that our protocols can achieve even better performance than our NTL based prototype implementations.

As mentioned before, an alternative approach to performing privacy preserving linear algebra operations consists in building circuits that compute each operation and evaluating them with a general purpose secure computation protocol [23, 22, 24]. In order to provide a baseline for the comparison of our specific purpose protocols with generic secure computation protocols, we analyse the performance of the online phase of the ABY framework [24] when evaluating a

Vector Length	π_{IP} NTL (s)	π_{IP} Optimized (s)	Trivial (s)	Overhead (%)
100	0.000531	0.000443	.000429	3.27
1000	0.001546	0.000506	0.000464	9.03
10000	0.010201	0.003061	0.001922	59.3
100000	0.096850	0.021670	0.012934	67.5
1000000	1.292219	0.139930	0.074516	87.8

Table 5: Execution times (in seconds) of NTL based and optimized implementations of protocol π_{IP} and a trivial protocol implementing the same functionality without security guarantees for different vector lengths. The overhead incurred by the optimized implementation of π_{IP} is calculated as the ratio between its execution time and that of the trivial protocol.

circuit that computes the inner product. To this end, we use the reference implementation of an arithmetic circuit that computes the inner product and its respective test application for the ABY framework, provided by its authors[1]. This implementation also considers vectors composed of 16 bits elements and uses only one thread. In Table 6, we present a comparison between the execution times of the optimized implementation of π_{IP} and of the online phase of ABY when evaluating the inner product circuit.

Vector Length	ABY [24] IP Circuit (s)	π_{IP} Optimized (s)
100	0.005598	0.000443
1000	0.478718	0.000506
2000	1.823921	0.001145
3000	3.843890	0.001261
4000	6.854674	0.001608

Table 6: Execution times (in seconds) of optimized implementation of π_{IP} and of the online phase of the ABY framework [24] when evaluating an arithmetic circuit that computes the inner product.

The data presented in Table 6 show that our specific purpose protocol enjoys performance gains in comparison to the ABY framework when evaluating the inner product functionality. However, one has to be careful and take these results within the right context. The ABY framework achieves generic secure computation of any circuit, while our protocols are restricted to their specific functionalities. On the other hand, our protocols achieve composable security against active adversaries, while the ABY framework is only secure in the passive case. Finally, ABY does not require a trusted initializer. If generic solutions secure against active adversaries were compared to our protocol, the performance gain would be more apparent because of the cost of generating/verifying message authentication codes to ensure security against malicious adversaries [23, 22, 24].

[1] `https://github.com/encryptogroup/ABY/tree/public/src/examples/innerproduct`, version of July 29, 2015.

9 Conclusion

We have proposed protocols for securely computing linear algebra problems in a two party scenario. Our solutions are efficient, universally composable and information-theoretically secure. We work in the commodity-based model, where the parties have available (as a setup assumption) pre-computed correlated data at the beginning of the protocol. This pre-computed data can be pre-distributed by a trusted authority (a trusted initializer) that need not engage furthermore in the remaining of the protocol. The data can also be pre-computed by the two players using computational assumptions. If this pre-computation step is universally composable (UC), the overall protocol will remain UC secure albeit no longer information-theoretically secure.

9.1 Future Works

In cases where the presence of a trusted initializer is undesirable (or infeasible), it would be an interesting sequel to this work to study how it can be substituted by a two-party protocol between Alice and Bob during the pre-processing phase. A simple but not optimized approach would be to compute multiplicative triples (a, b, c) such that $a \cdot b = c$ and reduce the TI behavior to computations over secure triples. In particular, such protocols can be constructed from additively homomorphic encryption schemes (*e.g.* Paillier [54]) through the techniques used in off-line phases of general multiparty computation protocols in the preprocessing model [23, 22] as outlined in [57]. One would have to be careful to preserve universal composability of the resulting protocol. The approach used in [23, 22] could be a way to achieve this.

Another approach consists in using oblivious transfer to compute the necessary multiplications [33]. Both approaches have been shown to achieve good performance, with OT based generation requiring only tens of miliseconds per triple. We refer the interested reader to [24, 57] for comprehensive surveys of the details and efficiency of different methods for generating multiplicative triples. The difficult part here would be to adapt the protocols proposed in [24] to the UC scenario. We remark, however, that the performance of the protocols presented in this paper is not affected by the efficiency of the trusted initializer, since its data can be precomputed independently from our protocol execution. In other words, the trusted initializer can compute its data at any given time before the actual inputs are given to our protocols, allowing for a large number of multiplicative triples to be precomputed and readily available to our protocols for future executions. Studying the scenarios here described are interesting sequels to this work.

References

[1] Arash Afshar, Zhangxiang Hu, Payman Mohassel, and Mike Rosulek. How to efficiently evaluate RAM programs with malicious security. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 702–729, Sofia, Bulgaria, April 26–30, 2015. Springer, Berlin, Germany. (Cited on page 2.)

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 673–701, Sofia, Bulgaria, April 26–30, 2015. Springer, Berlin, Germany. (Cited on page 2.)

[3] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual Symposium on Foundations of Computer Science*, pages 186–195, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press. (Cited on page 1.)

[4] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Berlin, Germany. (Cited on page 2.)

[5] Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 97–109, Santa Barbara, CA, USA, August 27–31, 1995. Springer, Berlin, Germany. (Cited on page 2, 6.)

[6] Donald Beaver. Commodity-based cryptography (extended abstract). In *29th Annual ACM Symposium on Theory of Computing*, pages 446–455, El Paso, Texas, USA, May 4–6, 1997. ACM Press. (Cited on page 2, 3, 6.)

[7] Donald Beaver. Server-assisted cryptography. In *Proceedings of the 1998 workshop on New security paradigms*, NSPW '98, pages 92–106, New York, NY, USA, 1998. ACM. (Cited on page 2, 6.)

[8] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany. (Cited on page 2, 5.)

[9] C. Blundo, B. Masucci, D. R. Stinson, and R. Wei. Constructions and bounds for unconditionally secure non-interactive commitment schemes. *Des. Codes Cryptography*, 26(1-3):97–110, June 2002. (Cited on page 6.)

[10] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008: 13th European Symposium on Research in Computer Security*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206, Málaga, Spain, October 6–8, 2008. Springer, Berlin, Germany. (Cited on page 1.)

[11] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 57–64, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer, Berlin, Germany. (Cited on page 1.)

[12] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: privacy-preserving aggregation of multi-domain network events and statistics. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, pages 15–15, Berkeley, CA, USA, 2010. USENIX Association. (Cited on page 1.)

[13] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press. (Cited on page 1, 7, 8, 14.)

[14] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany. (Cited on page 1.)

[15] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press. (Cited on page 1.)

[16] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Germany. (Cited on page 7.)

[17] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19, Chicago, Illinois, USA, May 2–4, 1988. ACM Press. (Cited on page 1, 2.)

[18] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 119–136, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany. (Cited on page 3, 5.)

[19] Ronald Cramer, Eike Kiltz, and Carles Padró. A note on secure computation of the Moore-Penrose pseudoinverse and its application to secure linear algebra. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 613–630, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Berlin, Germany. (Cited on page 3, 5.)

[20] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123, Santa Barbara, CA, USA, August 27–31, 1995. Springer, Berlin, Germany. (Cited on page 1.)

[21] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304, New York, NY, USA, March 4–7, 2006. Springer, Berlin, Germany. (Cited on page 1.)

[22] 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 Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18, Egham, UK, September 9–13, 2013. Springer, Berlin, Germany. (Cited on page 2, 5, 24, 25, 26.)

[23] Ivan Damgård, Valerio Pastro, Nigel P. 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, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Germany. (Cited on page 2, 5, 24, 25, 26.)

[24] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*. The Internet Society, 2015. (Cited on page 2, 4, 24, 25, 26.)

[25] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 164–181, Providence, RI, USA, March 28–30, 2011. Springer, Berlin, Germany. (Cited on page 1.)

[26] Rafael Dowsley, Jeroen van de Graaf, Davidson Marques, and Anderson C. A. Nascimento. A two-party protocol with trusted initializer for computing the inner product. In Yongwha Chung and Moti Yung, editors, *WISA 10: 11th International Workshop on Information Security Applications*, volume 6513 of *Lecture Notes in Computer Science*, pages 337–350, Jeju Island, Korea, August 24–26, 2011. Springer, Berlin, Germany. (Cited on page 3, 5, 8.)

[27] Rafael Dowsley, Jörn Müller-Quade, and Anderson C. A. Nascimento. On the possibility of universally composable commitments based on noisy channels. In André Luiz Moura dos Santos and Marinho Pilla Barcellos, editors, *Anais do VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, SBSEG 2008*, pages 103–114, Gramado, Brazil, September 1–5, 2008. Sociedade Brasileira de Computação (SBC). (Cited on page 1.)

[28] Rafael Dowsley, Jörn Müller-Quade, Akira Otsuka, Goichiro Hanaoka, Hideki Imai, and Anderson C. A. Nascimento. Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E94-A(2):725–734, 2011. (Cited on page 1, 3, 6.)

[29] Rafael Dowsley, Jeroen van de Graaf, Jörn Müller-Quade, and Anderson C. A. Nascimento. On the composability of statistically secure bit commitments. *Journal of Internet Technology*, 14(3):509–516, 2013. (Cited on page 1.)

[30] Wenliang Du. *A study of several specific secure two-party computation problems.* PhD thesis, West Lafayette, IN, USA, 2001. AAI3043719. (Cited on page 3, 15.)

[31] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology – CRYPTO'82*, pages 205–210, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA. (Cited on page 2.)

[32] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 537–556, Athens, Greece, May 26–30, 2013. Springer, Berlin, Germany. (Cited on page 2, 4.)

[33] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 116–129, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Germany. (Cited on page 26.)

[34] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On private scalar product computation for privacy-preserving data mining. In Choonsik Park and Seongtaek Chee, editors, *ICISC 04: 7th International Conference on Information Security and Cryptology*, volume 3506 of *Lecture Notes in Computer Science*, pages 104–120, Seoul, Korea, December 2–3, 2005. Springer, Berlin, Germany. (Cited on page 1, 3, 5.)

[35] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City,, New York, USA, May 25–27, 1987. ACM Press. (Cited on page 1, 2.)

[36] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 451–462, Chicago, Illinois, USA, October 4–8, 2010. ACM Press. (Cited on page 1.)

[37] Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 58–76, Cambridge, MA, USA, February 19–21, 2004. Springer, Berlin, Germany. (Cited on page 1.)

[38] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *In Proc. of the 5th Central European Conference on Cryptology MoraviaCrypt 2005*, 2005. (Cited on page 1.)

[39] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, TCC'13, pages 600–620, Berlin, Heidelberg, 2013. Springer-Verlag. (Cited on page 3.)

[40] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128, Barcelona, Spain, May 20–24, 2007. Springer, Berlin, Germany. (Cited on page 1.)

[41] Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 549–560, Berlin, Germany, November 4–8, 2013. ACM Press. (Cited on page 1, 2.)

[42] Florian Kerschbaum. Secure and sustainable benchmarking in clouds - A multi-party cloud application with an untrusted service provider. *Business & Information Systems Engineering*, 3(3):135–143, 2011. (Cited on page 1.)

[43] Florian Kerschbaum, Axel Schröpfer, Antonio Zilli, Richard Pibernik, Octavian Catrina, Sebastiaan de Hoogh, Berry Schoenmakers, Stelvio Cimato, and Ernesto Damiani. Secure collaborative supply-chain management. *IEEE Computer*, 44(9):38–43, 2011. (Cited on page 1.)

[44] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, New York, NY, USA, 1988. ACM. (Cited on page 1, 2.)

[45] Eike Kiltz, Gregor Leander, and John Malone-Lee. Secure computation of the mean and related statistics. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 283–302, Cambridge, MA, USA, February 10–12, 2005. Springer, Berlin, Germany. (Cited on page 1, 3.)

[46] Eike Kiltz, Payman Mohassel, Enav Weinreb, and Matthew K. Franklin. Secure linear algebra using linearly recurrent sequences. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 291–310, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Berlin, Germany. (Cited on page 3, 5.)

[47] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany. (Cited on page 2, 4.)

[48] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 476–494, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany. (Cited on page 2, 4.)

[49] Tal Moran and Gil Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 527–544, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany. (Cited on page 1.)

[50] Anderson C. A. Nascimento, Jörn Müller-Quade, Akira Otsuka, Goichiro Hanaoka, and Hideki Imai. Unconditionally secure homomorphic pre-distributed bit commitment and secure two-party computations. In Colin Boyd and Wenbo Mao, editors, *ISC 2003: 6th International Conference on Information Security*, volume 2851 of *Lecture Notes in Computer Science*, pages 151–164, Bristol, UK, October 1–3, 2003. Springer, Berlin, Germany. (Cited on page 3, 6.)

[51] Anderson C. A. Nascimento, Jörn Müller-Quade, Akira Otsuka, Goichiro Hanaoka, and Hideki Imai. Unconditionally non-interactive verifiable secret sharing secure against faulty majorities in the commodity based model. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS 04: 2nd International Conference on Applied Cryptography and Network Security*, volume 3089 of *Lecture Notes in Computer Science*, pages 355–368, Yellow Mountain, China, June 8–11, 2004. Springer, Berlin, Germany. (Cited on page 6.)

[52] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Germany. (Cited on page 2, 5.)

[53] Kobbi Nissim and Enav Weinreb. Communication efficient secure linear algebra. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 522–541, New York, NY, USA, March 4–7, 2006. Springer, Berlin, Germany. (Cited on page 3, 5.)

[54] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Berlin, Germany. (Cited on page 26.)

[55] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Berlin, Germany. (Cited on page 1.)

[56] Richard Pibernik, Yingying Zhang, Florian Kerschbaum, and Axel Schröpfer. Secure collaborative supply chain planning and inverse optimization - the JELS model. *European Journal of Operational Research*, 208(1):75–85, 2011. (Cited on page 1.)

[57] Pille Pullonen. Actively secure two-party computation: Efficient beaver triple generation. Master's thesis, University of Tartu, May 2013. (Cited on page 26.)

[58] Ronald L. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. Preprint available at http://people.csail.mit.edu/rivest/Rivest-commitment.pdf, 1999. (Cited on page 3, 6.)

[59] Thomas Schneider and Michael Zohner. GMW vs. Yao? efficient secure two-party computation with low depth circuits. In Ahmad-Reza Sadeghi, editor, *FC 2013: 17th International Conference on Financial Cryptography and Data Security*, volume 7859 of *Lecture Notes in Computer Science*, pages 275–292, Okinawa, Japan, April 1–5, 2013. Springer, Berlin, Germany. (Cited on page 2, 4.)

[60] Axel Schröpfer, Andreas Schaad, Florian Kerschbaum, Heiko Boehm, and Joerg Jooss. Secure benchmarking in the cloud. In Mauro Conti, Jaideep Vaidya, and Andreas Schaad, editors, *18th ACM Symposium on Access Control Models and Technologies, SACMAT '13, Amsterdam, The Netherlands, June 12-14, 2013*, pages 197–200. ACM, 2013. (Cited on page 1.)

[61] Rafael Tonicelli, Anderson C.A. Nascimento, Rafael Dowsley, Jörn Müller-Quade, Hideki Imai, Goichiro Hanaoka, and Akira Otsuka. Information-theoretically secure oblivious polynomial evaluation in the commodity-based model. *International Journal of Information Security*, pages 1–12, 2014. (Cited on page 3, 6.)

[62] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, Abhi Shelat, and Elaine Shi. SCORAM: Oblivious RAM for secure computation. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14: 21st Conference on Computer and Communications Security*, pages 191–202, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. (Cited on page 2.)

[63] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press. (Cited on page 1, 2.)