A Two-Party Protocol with Trusted Initializer for Computing the Inner Product

Rafael Dowsley[1], Jeroen van de Graaf[2], Davidson Marques[3], Anderson C. A. Nascimento[1]

[1] Department of Electrical Engineering, University of Brasília.
Campus Universitário Darcy Ribeiro,Brasília, CEP: 70910-900, Brazil,
Email: rafaeldowsley@redes.unb.br, andclay@ene.unb.br
[2] Department of Computer Science, Universidade Federal de Ouro Preto.
Ouro Preto, Minas Gerais, CEP: 35400-000, Brazil.
Email: jvdg@iceb.ufop.br
[3] Department of Computer Science, Universidade Federal de Minas Gerais,
Belo Horizonte, Minas Gerais, CEP 31270-901, Brazil.
E-mail: rodrigue@dcc.ufmg.br

Abstract. We propose the first protocol for securely computing the inner product modulo an integer m between two distrustful parties based on a trusted initializer, i.e. a trusted party that interacts with the players solely during a setup phase. We obtain a very simple protocol with universally composable security. As an application of our protocol, we obtain a solution for securely computing linear equations.

1 Introduction

Situations in which two parties wish to perform some joint computation arise naturally. Internet auctions are a well-known example. There are also many situations in which two organizations would like to compare their data bases, but privacy regulation, or the fact them being competitors, does not allow the simple solution: just sharing the data and do the computation.

There are many results showing that *any* function can be computed between two parties, by transforming the algorithm to a Boolean circuit and then emulating this circuit cryptographically [13]. Unfortunately, the cost of this emulation is prohibitively high, and the evaluation of even modest functions is often impractical. For this reason it is interesting to search for cryptographic primitives that efficiently implement the evaluation of some function f, in which f is not merely a Boolean, but some higher level primitive.

In this paper the primitive we study is computing the inner product modulo an arbitrary integer m. In other words, Alice provides an input vector $\overrightarrow{x}$, Bob provides an input vector $\overrightarrow{y}$, and they want to compute the inner product $\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle \bmod m = \sum_{i=1}^{n} x_i y_i \bmod m$. This is an interesting primitive since it is sufficiently high-level to have some immediate applications. It pops up in two-party protocols for linear algebra, statistical analysis, computational geometry and data mining. Additionally, we also

propose a protocol for securely solving linear systems based on our secure solution for computing the inner product.

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), either about the channel available between them (e.g. a noisy channel, oblivious transfer), or some trust assumption (e.g. the help of some trusted party or honest majority in a multiparty scenario). Here we study the inner product protocol under the assumption that a trusted initializer is available. This is a party who sends two different, but correlated, messages to the parties *before* the protocol starts, then he leaves. This model has appeared in many places in the literature, but was first formalized by Beaver in [1]

There are situations in which this model, with a trusted initializer, is realistic. For instance, suppose two hospitals want to do some combined statistical analysis on their data, but privacy legislation prohibits them to compare patient records directly. Conceivably, some third, neutral entity (say, the Association of All Hospitals) could help them to bootstrap the protocol in some session with all the three parties present. The third entity creates two DVDs with the correlated data, then, witnessed by the two others, wipes (or destroys) the platform that was used to create these DVDs, erasing all data.

In a somewhat looser trust model, we can suppose that the parties do not meet, but that parties buy their correlated data on the internet as originally proposed by Beaver in [1].

Observe that in the context of digital certificates such providers exist already. There they are called certification authorities.

Comparison with Other Work

We briefly relate our work to previous results stated in the literature.

Even tough the commodity based model (where parties receive cryptographic commodities/data from off-line servers before the protocol starts) was formalized just in [1], it has appeared in different flavors throughout the literature. For instance, in some sense, a Key Distribution Center can be interpreted as a trusted initializer for the task of private communication between two parties [7]. In [10], Rivest shows how to do Bit Commitment and Oblivious Transfer with a trusted initializer.

These protocols were generalized to oblivious polynomial evaluation and linear functional evaluation [11], verifiable secret sharing and two party computation of a multiplication in a field of characteristic p [9]. Another interesting primitive is oblivious linear function evaluation [8]. As far as we know, no work has been published on the inner product with a trusted initializer.

However, the inner product has been studied in other models. In fact, it attracted a lot of attention from the data mining community. See for instance the work of [3, 12] and of [5]. Unfortunately, many of the protocols proposed (including for other functions, not just for inner product) are seriously flawed, as already has been noted by others, see [5,6]. To repair the situation, Goethals et al. propose a protocol using homomomorphic encryption and provide a security proof [5]. We show that the commodity based framework provides a way to obtain an efficient, simple and elegant solution for

the secure two-party inner product problem and for solving linear equations. Moreover, we also show that it is possible to obtain this result in the strictest model of security available nowadays, that presented by the universal composability model [2].

Our Contributions

In this paper, we present a protocol for computing the inner product modulo any integer m with a trusted initializer, and prove it secure in the universal composability framework [2]. We restrict our analysis to the case of static adversaries.

We also show that the problems of solving linear equations securely can use the inner product protocol as a subprotocol:

Solving linear equation Alice input a matrix X and a vector $\overrightarrow{x}$, Bob inputs a matrix Y and a vector $\overrightarrow{y}$. They want the vector $\overrightarrow{z}$ that solves the equation $(X+Y)\cdot\overrightarrow{z} = \overrightarrow{x}+\overrightarrow{y}$

It should be emphasized that this last protocols can only be proven secure assuming that we use some finite field $\mathbb{F}$ of cardinality $m = p^k$. The reason is that solving linear equations requires the ability to compute inverse elements, which are only well-defined over fields. In addition, our techniques are only able to suitable for *finite* fields.

The Thesis of Du

An important starting point for this research was the Ph.D. thesis of Du. Here, a strong point is made for the need of practical two-party computations, with several interesting applications to privacy-preserving data mining and statistical computing.

It quickly became obvious that the thesis leaves much to be desired from a theoretical point of view: no formal model is presented, and no formal security proofs are given. However, it took several months to realize that the situation was much more serious.

In the first place, often it is not defined to which set the values used in the protocols belong. Almost always there are several auxiliary variables that take on a random value, i.e. that are randomly chosen. However, in most cases the underlying set is assumed to be $\mathbb{Z}$, which does not allow a uniform distribution. For this reason, any attempt to prove security fails and the protocols are insecure. In some situations this can be remedied choosing a large modulus M and performing the computations mod M. For instance, the inner product protocol can be modified in this way, and the result is secure.

In the second place, Du proposes a division protocol in $\mathbb{Z}$ that is not secure. This was shown in [6], which presents another, much more elaborate, division protocol. One can think: no problem, we will do our computations in a finite field. But that almost never resolve the problem. $\mathbb{Z}$ and $\mathbb{Q}$ are well-ordered sets, but finite fields are not. For instance, in $\mathbb{Q}$ one knows that $x < y$ implies $1/x > 1/y$, but in a finite field this does not make sense, they have no notion of small and large.

In the case of the protocol for solving a linear equation over a finite field, it *does* make sense to consider this question over $\mathbb{F}_q$. But it is not clear how to extend this protocol to $\mathbb{Z}$ or $\mathbb{Q}$. Here, choosing a large (prime) modulus does not work. This reduces severely the applicability of this protocol. The situation for several other protocols is worse. For instance, restricting the protocol for linear programming to $\mathbb{F}_q$ does not make sense, and in $\mathbb{Z}$ the protocol proposed is hopelessly insecure.

2 The Universal Composability Framework

The objective of the UC framework is to allow the truly modular design of cryptographic protocols. The crux of the UC framework is the introduction of an environment, $\mathcal{Z}$, who supplies input to the parties, Alice, Bob and the Adversary, $\mathcal{A}$, and receives there outputs.

In order to prove security of a specific protocol implementation, we will compare it with some idealized version of the protocol and show that $\mathcal{Z}$ cannot tell the difference. This idealized version is called the (ideal) *functionality*, and can be seen as some black box that does exactly what the protocol is supposed to do, i.e. follow the protocol specification faithfully. Observe that the functionality must also deal with invalid inputs or other unexpected behavior from the parties, because even in the idealized model we have to assume that parties can be corrupted and behave dishonestly, by changing their inputs, for instance.

The whole purpose of the UC security definition is to avoid the existence of attacks possible in the real protocol implementation that do not exist in the ideal model. In other words, we want that each attack against the real protocol corresponds to an attack against the ideal protocol. This proves that the real protocol is at least as secure as the ideal functionality, so, assuming the latter is well-defined, we obtain a secure protocol implementation.

More precisely, we want to show that for every adversary in the real protocol, denoted $\mathcal{A}$, there exists an ideal adversary in the ideal protocol, denotes $\mathcal{S}$, such that no environment exists that can tell the difference:

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}\ : \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \equiv \text{IDEAL}_{\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle},\mathcal{S},\mathcal{Z}}$$

In this expression REAL stands for $\mathcal{Z}$'s output, defined as a single bit, after observing the real Alice and Bob running the protocol imlementation with real adversary $\mathcal{A}$, whereas IDEAL stands for $\mathcal{Z}$'s single bit output, after observing the ideal Alice and Bob running the functionality with ideal adversary $\mathcal{S}$. The probability distribution is taken over all the parties' random tapes. Observe that $\mathcal{Z}$ acts as a distinguisher, trying to tell the real and ideal model apart. The universal quantifier guarantees that this is impossible. Indistinguishability between families of probability distributions comes in various flavors: perfect, statistical and computational indistinguisability. In this paper we will only use perfect indistinguisability, meaning that the two distributions are identical.

3 The Inner Product Protocol

3.1 Protocol Specification and Notation

We briefly describe the players inputs and outputs specified by the protocol. In the following, we denote by $\mathbb{Z}_m$ the set $\{1, \cdots, m-1\}$, by $\mathbb{Z}_m^n$ the space of all n-tuples of elements of $\mathbb{Z}_m$. The act of randomly choosing an element $\overrightarrow{x}$ from $\mathbb{Z}_m^n$ is represented by $\overrightarrow{x} \in_R \mathbb{Z}_m^n$. The players are conveniently named Alice and Bob, while the trusted initializer is called TI.

	Alice	Bob
input	$\overrightarrow{x} \in \mathbb{Z}_m^n$	$\overrightarrow{y} \in \mathbb{Z}_m^n$
output	$r \in \mathbb{Z}_m$	$\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle - r \in \mathbb{Z}_m$

3.2 Ideal Functionality for the Inner Product

We describe the ideal functionality for securely computing the inner product.

- Upon receiving an (ASENDSINPUT, $\overrightarrow{x}$, *sid*) message from A:
Ignore any subsequent ASENDSINPUT messages. If $\overrightarrow{x} \notin \mathbb{Z}_m^n$, then send an INVALIDINPUT message to A and B and stop. If no BSENDSINPUT message has been received from B, then store $\overrightarrow{x}$ and *sid*, and send the public delayed output (AINPUTRECEIVED, *sid*) to B; else choose $r \in_R \mathbb{Z}_m$, set $u := r$ and $v := \langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle - r$, send the public delayed outputs (AGETSOUTPUT, u, *sid*) to A and (BGETSOUTPUT, v, *sid*) to B.

- Upon receiving a (BSENDSINPUT, $\overrightarrow{y}$, *sid*) message from B:
Ignore any subsequent BSENDSINPUT messages. If $\overrightarrow{y} \notin \mathbb{Z}_m^n$, then send an INVALIDINPUT message to A and B and stop. If no ASENDSINPUT message has been received from A, then store $\overrightarrow{y}$ and *sid*, and send the public delayed output (BINPUTRECEIVED, *sid*) to A; else choose $r \in_R \mathbb{Z}_m$, set $u := r$ and $v := \langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle - r$, send the public delayed outputs (AGETSOUTPUT, u, *sid*) to A and (BGETSOUTPUT, v, *sid*) to B.

It is interesting to observe that, depending on the security parameters, the ideal functionality might leak a lot of information on the players inputs. For instance, if n, the size of the vectors) is very low, and m, the modulus is very high, both parties can deduce the other's input. On the other hand, for $m = 2$ the protocol only gives the parity of the inner product, which is already meaningful for $n = 1$ in which case it reduces to the matchmaking problem.

3.3 Trusted Initializer Functionality

We also define an ideal functionality modeling the behavior of the trusted initializer.

- When first activated, choose $\overrightarrow{x_0} \in_R \mathbb{Z}_m^n$, $\overrightarrow{y_0} \in_R \mathbb{Z}_m^n$, compute $s_0 := \langle \overrightarrow{x_0} \cdot \overrightarrow{y_0} \rangle$ and distribute $\overrightarrow{x_0}$ to Alice and $(\overrightarrow{y_0}, s_0)$ to Bob.

3.4 Protocol implementation

	Alice	Bob
TI	$\overrightarrow{x_0} \in_R \mathbb{Z}_m^n$	$\overrightarrow{y_0} \in_R \mathbb{Z}_m^n$; $s_0 := \langle \overrightarrow{x_0} \cdot \overrightarrow{y_0} \rangle$
data sent by TI	$\mu_A := \overrightarrow{x_0}$	$\mu_B := (\overrightarrow{y_0}, s_0)$
input	$\overrightarrow{x} \in \mathbb{Z}_m^n$	$\overrightarrow{y} \in \mathbb{Z}_m^n$
protocol		(1) $\overrightarrow{y_1} := \overrightarrow{y} - \overrightarrow{y_0} \in \mathbb{Z}_m^n$ Send $\mu_1 := (\overrightarrow{y_1})$ to Alice
	(2) If μ_1 is invalid then abort $\overrightarrow{x_1} := \overrightarrow{x} + \overrightarrow{x_0} \in \mathbb{Z}_m^n$ $r \in_R \mathbb{Z}_m$ $r_1 := \langle \overrightarrow{x} \cdot \overrightarrow{y_1} \rangle - r$ Send $\mu_2 := (\overrightarrow{x_1}, r_1)$ to Bob	
		If μ_2 is invalid then abort
output	$u := r$	$v := \langle \overrightarrow{x_1} \cdot \overrightarrow{y_0} \rangle + r_1 - s_0$

It is straightforward to check the correctness of the protocol.

$$
\begin{aligned}
v &:= \langle \overrightarrow{x_1} \cdot \overrightarrow{y_0} \rangle + r_1 - s_0 \\
&= \langle (\overrightarrow{x} + \overrightarrow{x_0}) \cdot \overrightarrow{y_0} \rangle + (\langle \overrightarrow{x} \cdot (\overrightarrow{y} - \overrightarrow{y_0}) \rangle - r) - \langle \overrightarrow{x_0} \cdot \overrightarrow{y_0} \rangle \\
&= \langle \overrightarrow{x} \cdot \overrightarrow{y_0} \rangle + \langle \overrightarrow{x_0} \cdot \overrightarrow{y_0} \rangle + \langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle - \langle \overrightarrow{x} \cdot \overrightarrow{y_0} \rangle - r - \langle \overrightarrow{x_0} \cdot \overrightarrow{y_0} \rangle \\
&= \langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle - r
\end{aligned}
$$

4 Security Proof

For clarity we write Alice and Bob in the proofs, to avoid confusion with the adversary $\mathcal{A}$. $\mathcal{S}$ runs $\mathtt{A}'$ internally. We will also denote all variables in the simulated environment with a prime $'$.

4.1 Alice Corrupted, Bob Honest

Simulation $\mathcal{S}$ runs an internal (embedded) copy of $\mathcal{A}$ called $\mathtt{A}'$. Observe that Alice is corrupted, so $\mathcal{S}$ has access to Alice's input $\overrightarrow{x}$. The interactions of $\mathtt{A}'$ with $\mathcal{S}$ are those of Alice in the real protocol with the other parties, $\mathcal{Z}$, the TI, and Bob. We now show how $\mathcal{S}$ acts when such interactions take place:

# Events	$\mathcal{S}$'s actions
1 Onset of the Simulation	Choose $\overrightarrow{x_0}' \in_R \mathbb{Z}_m^n$ and send $\mu'_A := \overrightarrow{x_0}'$ to $\mathtt{A}'$
2 Get input $\overrightarrow{x}$	$\mathcal{S}$ forwards $\overrightarrow{x}$ from $\mathcal{Z}$, i.e. $\overrightarrow{x}' := \overrightarrow{x}$
3 Message (BINPUTRECEIVED, *sid*)	$\mathcal{S}$ feeds $\mu'_1 := \overrightarrow{y_1}' \in_R \mathbb{Z}_m^n$ to $\mathtt{A}'$
4 Message μ'_2	If $\mathtt{A}'$ sends $(\overrightarrow{x_1}', r'_1)$ as message μ'_2, then $\mathcal{S}$ sends a message (ASENDSINPUT, $\overrightarrow{x}$, *sid*) to $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$. If $\mathtt{A}'$ sends something invalid in μ'_2, then $\mathcal{S}$ sends something invalid to $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$.
5 Output u';	As long as no reponse from $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$ is received, $\mathcal{S}$ does nothing, even if this means waiting forever. When an INVALIDINPUT message is received, $\mathcal{S}$ forwards it to $\mathcal{Z}$. When an AGETSOUTPUT message is received, $\mathcal{S}$ does the following: It lets the functionality deliver the message (BGETSOUTPUT, v, *sid*). $\mathcal{S}$ also intercepts the simulated output $u' = r'$ and verifies if this value is consistent with the input $\overrightarrow{x}$, and the simulated messages μ'_A, μ'_1 and μ'_2. If consistent, $\mathcal{S}$ substitutes the simulated output u' for the real output u obtained from $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$ by setting $u_{\mathcal{S}} = u$; else it sets $u_{\mathcal{S}} = u'$. Then $u_{\mathcal{S}}$ is sent to $\mathcal{Z}$ through Alice's interface.

Indistinguishability 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 $\mathtt{A}'$'s strategy is, it either sends a valid or an invalid message μ'_2. If the message is invalid, both the simulated and the ideal protocol will send INVALIDINPUT to the two parties, which will be forwarded to $\mathcal{Z}$.
3. Even if $\mathtt{A}'$ sent a valid message μ'_2, 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 $\overrightarrow{x}' = \overrightarrow{x}$ and the messages μ'_A, μ'_1 and μ'_2. In the case $\mathtt{A}'$ did deviate, $\mathcal{S}$ detects this and does nothing, i.e. it forwards the output produced by $\mathtt{A}'$ directly to $\mathcal{Z}$.
4. In the case $\mathtt{A}'$ did follow the protocol, $\mathcal{S}$ substitutes $\mathtt{A}'$'s output u_s for the output $u_{\mathcal{F}}$ obtained from $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$.

If we consider $\mathtt{A}'$ (and $\mathcal{A}$) as deterministic algorithms whose probabilism comes from a random tape, it follows that $\mathtt{A}'$ behavior is completely determined by these random bits, called s_A, the incoming message μ'_A, the input $\overrightarrow{x}$ and the incoming message μ'_1. We already know that the random bits s_A and the input $\overrightarrow{x}$ have the same distribution in the real and ideal protocol, because of the way the model is defined.

So in order to show that $\forall \mathcal{A}\, \exists \mathcal{S}\, \forall \mathcal{Z}\ : \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \equiv \text{IDEAL}_{\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle},\mathcal{S},\mathcal{Z}}$, it suffices to show that the incoming message μ'_A produced by $\mathcal{S}$ has the same distribution as the incoming message μ_A produced by the TI in the real protocol. But this is trivial, since both are generated from the same distribution, the uniform distribution on $\mathbb{Z}_m^n$.

In addition, we must show that μ'_1 produced by $\mathcal{S}$ and μ_1 sent by Bob have the same distribution. Observe that $\mu_1 := \overrightarrow{y_1} := \overrightarrow{y} - \overrightarrow{y_0}$, with $\overrightarrow{y_0} \in_R \mathbb{Z}_m^n$. Since both TI and Bob are honest in the real protocol, it follows that both μ'_1 and μ_1 are generated according to the uniform distribution on $\mathbb{Z}_m^n$.

So we conclude that $\mathtt{A}'$'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.

4.2 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 $\mathtt{B}'$. Observe that Bob is corrupted, so $\mathcal{S}$ has access to Bob's input $\overrightarrow{y}$. The interactions of $\mathtt{B}'$ with $\mathcal{S}$ are those of Bob in the real protocol with the other parties, $\mathcal{Z}$, the TI, and Alice. We give an overview how $\mathcal{S}$ acts

# Events	$\mathcal{S}$'s actions
1 Onset of the Simulation	$\mathcal{S}$ sets $\overrightarrow{y_0}' \in_R \mathbb{Z}_m^n$, $s'_0 \in_R \mathbb{Z}_m$ and feeds $\mu'_B := (\overrightarrow{y_0}', s'_0)$ to $\mathtt{B}'$
2 Get input $\overrightarrow{y}$;	$\mathcal{S}$ forwards $\overrightarrow{y}$ from $\mathcal{Z}$, i.e. $\overrightarrow{y}' := \overrightarrow{y}$
3 Message μ'_1	If $\mathtt{B}'$ sends $\overrightarrow{y_1}'$ as message μ'_1, then $\mathcal{S}$ sends a message (BSENDSINPUT, $\overrightarrow{y}$, *bsid*) to $\mathcal{F}_{\langle \overrightarrow{x}, \overrightarrow{y} \rangle}$. If $\mathtt{B}'$ sends something invalid in μ'_1, then $\mathcal{S}$ sends something invalid to $\mathcal{F}_{\langle \overrightarrow{x}, \overrightarrow{y} \rangle}$. $\mathcal{S}$ lets the functionality deliver the message (BINPUTRECEIVED, *sid*)
4 Message (BGETSOUTPUT, v, *sid*)	$\mathcal{S}$ sets $(\overrightarrow{x_1}' \in_R \mathbb{Z}_m^n, r'_1 \in_R \mathbb{Z}_m^n$ and feeds $\mu'_2 := (\overrightarrow{x_1}', r'_1)$ to $\mathtt{B}'$; $\mathcal{S}$ lets the functionality deliver the message (AGETSOUTPUT, u, *sid*) to A. $\mathcal{S}$ intercepts the simulated output $v' = \langle \overrightarrow{x_1}' \cdot \overrightarrow{y_0}' \rangle + r'_1 - s'_0$ and verifies if this value is consistent with the input $\overrightarrow{y}$, and the simulated messages μ'_B, μ'_1 and μ'_2. If consistent, $\mathcal{S}$ substitutes the simulated output v' with the real output v obtained from $\mathcal{F}_{\langle \overrightarrow{x}, \overrightarrow{y} \rangle}$ by setting $v_{\mathcal{S}} = v$; else it sets $v_{\mathcal{S}} = v'$. As long as no v' from $\mathtt{B}'$ is received, $\mathcal{S}$ does not forward the message BGETSOUTPUT, even if this means waiting forever. Then $v_{\mathcal{S}}$ is sent to $\mathcal{Z}$ through Bob's interface.

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

4.3 Alice and Bob Honest

If neither party is corrupted, $\mathcal{A}$ simulates the Trusted Initializer Functionality, sees a transcript of the message sent between Alice and Bob and let the functionality send the public delayed outputs in the ideal protocol when $\mathcal{A}$ deliver the respective messages in the simulated execution.

4.4 Alice and Bob Corrupted

The protocol is a deterministic function of the parties' inputs and random tapes. When both Alice and Bob are corrupted, $\mathcal{S}$ has full access to this information and can simulate perfectly.

5 Solving Linear Equations

We now show how to use the previously proposed protocol for computing the inner product in order to obtain a new protocol for securely solving linear equations. Before we introduce the ideal functionalities related to this task, we briefly introduce the notation used in the protocol description.

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 $\mathbb{F}_q$. The act of randomly choosing an element $\overrightarrow{x}$ from $\mathbb{F}_q^n$ is represented by $\overrightarrow{x} \in_R \mathbb{F}_q^n$. $\mathbb{F}_q^{n \times n}$ represents the space of all $n \times n$ matrices with elements belonging to $\mathbb{F}_q$, while $SL(\mathbb{F}_q)$ represents the set of all non-singular $n \times n$ matrices with elements belonging to $\mathbb{F}_q$. Sum, multiplication and multiplication by scalar for vectors and matrices are defined as usual.

5.1 Ideal Functionality

- Upon receiving an (ASENDSINPUT, $\overrightarrow{x}$, X, *sid*) message from A:
Ignore any subsequent ASENDSINPUT messages. If $\overrightarrow{x} \notin \mathbb{Z}_m^n$ or $X \notin SL(\mathbb{F}_q)$, then send an INVALIDINPUT message to A and B and stop. If no BSENDSINPUT message has been received from B, then store $\overrightarrow{x}$, X and *sid*, and send the public delayed output (AINPUTRECEIVED, *sid*) to B; else find $\overrightarrow{z}$ such that $(X+Y)\overrightarrow{z} = \overrightarrow{x} + \overrightarrow{y}$ and send the public delayed output (BGETSOUTPUT, $\overrightarrow{z}$, *sid*) to B.

- Upon receiving a (BSENDSINPUT, $\overrightarrow{y}$, Y, *sid*) message from B:
Ignore any subsequent BSENDSINPUT messages. If $\overrightarrow{y} \notin \mathbb{Z}_m^n$ or $Y \notin SL(\mathbb{F}_q)$, then send an INVALIDINPUT message to A and B and stop. If no ASENDSINPUT message has been received from A, then store $\overrightarrow{y}$, Y and *sid*, and send the public delayed output (BINPUTRECEIVED, *sid*) to A; else find $\overrightarrow{z}$ such that $(X+Y)\overrightarrow{z} = \overrightarrow{x} + \overrightarrow{y}$ and send the public delayed output (BGETSOUTPUT, $\overrightarrow{z}$, *sid*) to B.

5.2 Protocol implementation

Our approach is based on Du's approach [3]. The solution $\overrightarrow{z}$ to the linear equation $(X+Y)\overrightarrow{z} = \overrightarrow{x} + \overrightarrow{y}$ is equal to the solution $\overrightarrow{z}$ in $P(X+Y)QQ^{-1}\overrightarrow{z} = P(\overrightarrow{x} + \overrightarrow{y})$, 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 $P(X+Y)Q\overrightarrow{t} = P(\overrightarrow{x} + \overrightarrow{y})$, so $t = Q^{-1}\overrightarrow{z}$. In other words, the solution that Alice gets to see is the final solution $\overrightarrow{z}$, blinded with a random invertible matrix Q^{-1}. To allow Alice to compute $P(X+Y)Q$ and

$P(\overrightarrow{x}+\overrightarrow{y})$ without her learning Y or y, we use the inner product as a subprotocol, since matrix multiplication is nothing but the inner products of the right rows and columns.

Note that we can do all subprotocols in parallel since we have proven it UC. Note also that we can modify the previous protocol (without affecting its security) so that the value r can be is chosen randomly by TI and pre-distributed. Though the protocol notation below seems to suggest otherwise, it should be pointed out that the initialization phase for the main protocol and the subprotocols takes place at the same time. The Trusted Initializer Functionality is similar to the previous one, but in addition to the data used by the inner product protocols, it also pre-distributes the other data needed by the protocol below.

In the following, $(R;[PX-R]) := \Pi_{\langle \overrightarrow{x}\cdot\overrightarrow{y}\rangle}(X;P)$ denotes a protocol where Alice inputs the matrix X, Bob inputs the matrix P, Alice receives a random R as output while Bob receives $PX-R$. As stated previously, such a protocol for secure multiplication of matrices is clealry implementable given our protocol for computing the inner product.

Trusted Initializer	$Q, R \in_R SL(\mathbb{F}_q)$ $U \in_R \mathbb{F}_q^{n\times n}$ $\overrightarrow{s} \in_R \mathbb{F}_q^n$	
Data sent by TI	$\mu_A := (R, V = RQ+U, \overrightarrow{s})$	$\mu_B := (Q, U)$
	Alice	**Bob**
Input	$X \in SL(\mathbb{F}_q), \overrightarrow{x} \in \mathbb{F}_q^n$	$Y \in SL(\mathbb{F}_q), \overrightarrow{y} \in \mathbb{F}_q^n$
Step 1:		$P \in_R SL(\mathbb{F}_q)$
Call subprotocol $\Pi_{\langle \overrightarrow{x}\cdot\overrightarrow{y}\rangle}$ Call subprotocol $\Pi_{\langle \overrightarrow{x}\cdot\overrightarrow{y}\rangle}$	$(R;[PX-R]) := \Pi_{\langle \overrightarrow{x}\cdot\overrightarrow{y}\rangle}(X;P)$ $(\overrightarrow{s};[P\overrightarrow{x}-\overrightarrow{s}]) := \Pi_{\langle \overrightarrow{x}\cdot\overrightarrow{y}\rangle}(\overrightarrow{x};P)$	
		$M := [PX-R]Q+PYQ-U$ $\overrightarrow{c} := [P\overrightarrow{x}-\overrightarrow{s}]+P\overrightarrow{y}$ Send $\mu_1 := (M, \overrightarrow{c})$ to Alice
Step 2:	If μ_1 is invalid then abort $N := M+V$ $\overrightarrow{d} := \overrightarrow{c}+\overrightarrow{s}$ Find $\overrightarrow{t}$ such that $N\overrightarrow{t} = \overrightarrow{d}$ Send $\mu_2 := (\overrightarrow{t})$ to Bob	
Step 3:		Compute $\overrightarrow{z} = Q^{-1}\overrightarrow{t}$
Output	$u := \varepsilon$	$v := \overrightarrow{z}$

The correctness of the protocol is trivially verified:

$N := M + V = [PX - R]Q + PYQ - U + RQ + U = P(X + Y)Q$
$\overrightarrow{d} := \overrightarrow{c} + \overrightarrow{s} = [P\overrightarrow{x} - \overrightarrow{s}] + P\overrightarrow{y} + \overrightarrow{s} = P(\overrightarrow{x} + \overrightarrow{y})$
so Alice solves the equaltion $P(X + Y)Q\overrightarrow{t} = P(\overrightarrow{x} + \overrightarrow{y})$.
In order to find the solution to $(X + Y)\overrightarrow{z} = \overrightarrow{x} + \overrightarrow{y}$, Bob must compute $\overrightarrow{z} = Q^{-1}\overrightarrow{t}$.

6 Security Proof

6.1 Alice Corrupted, Bob Honest

Simulation $\mathcal{S}$ runs an internal (embedded) copy of $\mathcal{A}$ called $\mathtt{A}'$. Observe that Alice is corrupted, so $\mathcal{S}$ has access to Alice's inputs $\overrightarrow{x}$ and X. The interactions of $\mathtt{A}'$ with $\mathcal{S}$ are those of Alice in the real protocol with the other parties, $\mathcal{Z}$, the TI, and Bob. We now show how $\mathcal{S}$ acts when such interactions take place (the simulation deals with the inner products in the same way as explained previously, and so these steps will be omitted):

# Events	$\mathcal{S}$'s actions
1 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 $\mathtt{A}'$.
2 Get input $\overrightarrow{x}$ and X	$\mathcal{S}$ forwards $\overrightarrow{x}$ and X from $\mathcal{Z}$, i.e. $\overrightarrow{x}' := \overrightarrow{x}$ and $X' = X$. $\mathcal{S}$ also chooses $\overrightarrow{y}' \in_R \mathbb{F}_q^n$, $Y' \in_R SL(\mathbb{F}_q)$ and $P' \in_R SL(\mathbb{F}_q)$.
3 Message (BINPUTRECEIVED, *sid*)	$\mathcal{S}$ computes M' and $\overrightarrow{c}'$, and sends $\mu_1' := (M', \overrightarrow{c}')$ to $\mathtt{A}'$
4 Message μ_2'	If $\mathtt{A}'$ sends $\overrightarrow{t}'$ as message μ_2', then $\mathcal{S}$ sends a message (ASENDSINPUT, $\overrightarrow{x}$, X, *sid*) to $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$. If $\mathtt{A}'$ sends something invalid in μ_2', then $\mathcal{S}$ sends something invalid to $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$.
5 Message (BGETSOUTPUT, $\overrightarrow{z}$, *sid*)	$\mathcal{S}$ lets the functionality deliver the message (BGETSOUTPUT, v *sid*).

Indistinguishability 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 $\mathtt{A}'$'s strategy is, it either sends a valid or an invalid message μ_2'. If the message is invalid, both the simulated and the ideal protocol will send INVALIDINPUT to the two parties, which will be forwarded to $\mathcal{Z}$.

If we consider $\mathtt{A}'$ (and $\mathcal{A}$) as deterministic algorithms whose probabilism comes from a random tape, it follows that $\mathtt{A}'$ behavior is completely determined by these random bits, called s_A, the pre-distributed data, the inputs $\overrightarrow{x}$ and X, and the incoming message μ_1'. We already know that the random bits s_A and the inputs $\overrightarrow{x}$ and 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 $\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}\ : \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \equiv \text{IDEAL}_{\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}, \mathcal{S}, \mathcal{Z}}$, 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 μ_1' produced by $\mathcal{S}$ and μ_1 sent by Bob have the same distribution. Observe that $M := [PX - R]Q + PYQ - U$ with $U \in_R \mathbb{F}_q^{n \times n}$ and $\overrightarrow{c} := [P\overrightarrow{x} - \overrightarrow{s}] + P\overrightarrow{y}$ with $P \in_R SL(\mathbb{F}_q)$. Since both TI and Bob are honest in the real protocol, it follows that both μ_1' and μ_1 are generated according to the uniform distribution.

So we conclude that $\mathtt{A}'$'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.

6.2 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 $\mathtt{B}'$. Observe that Bob is corrupted, so $\mathcal{S}$ has access to Bob's input $\overrightarrow{y}$. The interactions of $\mathtt{B}'$ with $\mathcal{S}$ are those of Bob in the real protocol with the other parties, $\mathcal{Z}$, the TI, and Alice. We give an overview how $\mathcal{S}$ acts

# Events	$\mathcal{S}$'s actions
1 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 $\mathtt{B}'$.
2 Get input $\overrightarrow{y}$ and Y	$\mathcal{S}$ forwards $\overrightarrow{y}$ and Y from $\mathcal{Z}$, i.e. $\overrightarrow{y}' := \overrightarrow{y}$ and $Y' = Y$. $\mathcal{S}$ also chooses $\overrightarrow{x}' \in_R \mathbb{F}_q^n$ and $X' \in_R SL(\mathbb{F}_q)$.
3 Message μ_1'	If $\mathtt{B}'$ sends $(M', \overrightarrow{c}')$ as message μ_1', then $\mathcal{S}$ sends a message (BSENDSINPUT, $\overrightarrow{y}$, Y, *bsid*) to $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$. If $\mathtt{B}'$ sends something invalid in μ_1', then $\mathcal{S}$ sends something invalid to $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$. $\mathcal{S}$ lets the functionality deliver the message (BINPUTRECEIVED, *sid*)
4 Message (BGETSOUTPUT, v, *sid*)	$\mathcal{S}$ computes N', $\overrightarrow{y}'$, finds $\overrightarrow{t}'$ and sends $\mu_2' := (\overrightarrow{t}')$ to $\mathtt{B}'$; $\mathcal{S}$ intercepts the simulated output v' 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 v' with the real output v obtained from $\mathcal{F}_{\langle \overrightarrow{x} \cdot \overrightarrow{y} \rangle}$ by setting $v_{\mathcal{S}} = v$; else it sets $v_{\mathcal{S}} = v'$. As long as no v' from $\mathtt{B}'$ is received, $\mathcal{S}$ does not forward the message BGETSOUTPUT, even if this means waiting forever. Then $v_{\mathcal{S}}$ is sent to $\mathcal{Z}$ through Bob's interface.

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

6.3 Alice and Bob Honest

If neither party is corrupted, $\mathcal{A}$ simulates the Trusted Initializer Functionality, sees a transcript of the message sent between Alice and Bob and let the functionality send the public delayed outputs in the ideal protocol when $\mathcal{A}$ deliver the respective messages in the simulated execution.

6.4 Alice and Bob Corrupted

The protocol is a deterministic function of the parties' inputs and random tapes. When both Alice and Bob are corrupted, $\mathcal{S}$ has full access to this information and can simulate perfectly.

7 Conclusions

We have presented a protocol for computing the inner product between two vectors defined over $\mathbb{Z}_m$. We proved our solution secure in the UC framework. As an application of our protocol, we built on top of it another protocol that securely solves linear equations over finite fields.

We believe that the commodity/ trusted initializer model, where parties receive predistributed data during a setup phase from a trusted source and then go on and proceed with their secure computations without further interaction with this trusted source, provides a practical and interesting framework for secure two-party computations. The solutions obtained usually demand a large (but not prohibitively with today's technology) storage capability, but are otherwise very efficient from a computational point of view.

We hope our protocols presented here are used to provide more complex tasks, such as secure data mining and so on. Generalizations for approximately solving linear equations over the reals are also an interesting sequel to this work [4].

References

1. Donald Beaver. Commodity-based cryptography (extended abstract). In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 446–455, New York, NY, EUA, 1997. ACM.
2. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *focs*, page 136. Published by the IEEE Computer Society, 2001.
3. Wenliang Du. *A study of several specific secure two-party computation problems*. PhD thesis, Purdue University, West-Lafayette, Indiana, 2001.
4. J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. *Automata, Languages and Programming*, pages 927–938.

5. Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *Information Security and Cryptology - ICISC 2004, 7th International Conference, Seoul, Korea, December 2-3, 2004, Revised Selected Papers*, volume 3506 of *Lecture Notes in Computer Science*, pages 104–120. Springer, 2004.
6. Eike Kiltz, Gregor Leander, and John Malone-Lee. Secure computation of the mean and related statistics. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, EUA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 283–302. Springer, 2005.
7. Tsutomu Matsumoto and Hideki Imai. On the key predistribution system: A practical solution to the key distribution problem. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, EUA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 185–193. Springer, 1987.
8. R. Meier, B. Przydatek, and J. Wullschleger. Robuster combiners for oblivious transfer. *Theory of Cryptography*, pages 404–418.
9. 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 *Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings*, volume 3089 of *Lecture Notes in Computer Science*, pages 355–368. Springer, 2004.
10. Ron Rivest. Unconditionally Secure Commitment and Oblivious Transfer Schemes Using Private Channels and a Trusted Initializer. http://people.csail.mit.edu/rivest/Rivest-commitment.pdf, 1999.
11. 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. Cryptology ePrint Archive, Report 2009/270, 2009.
12. Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 639–644. ACM, 2002.
13. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, 3-5 November 1982, Chicago, Illinois, EUA*, pages 160–164. IEEE, 1982.