Universally Composable Committed Oblivious Transfer With A Trusted Initializer

Adriana C. B. Pinto [1], Bernardo Machado David [1] Jeroen van de Graaf[2], Anderson C. A. Nascimento[1]

[1]Departmento de Engenharia Elétrica – Universidade de Brasília (UNB)
Campus Universitário Darcy Ribeiro – 70910-900 – Brasília – DF – Brasil

[2]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Av. Antônio Carlos 6627 – 31270-901 – Belo Horizonte (MG) – Brasil

adrianacbp@unb.br, bernardo.david@aluno.unb.br

jvdg@dcc.ufmg.br, andclay@ene.unb.br

Abstract. *Committed Oblivious Transfer (COT) is a two-party primitive that combines one-out-of-two oblivious transfer with bit commitment. In the beginning of COT, a sender is committed to bits b_0,b_1 and a receiver to a choice bit c. In the end, the receiver is committed to b_c without learning anything about b_{1-c}, while the sender learns nothing about c. This primitive implies secure multi-party computation assuming that a broadcast channel is available. In this paper, we introduce the first universally composable unconditionally secure committed oblivious transfer protocol based on a Trusted Initializer (TI), which pre-distributes data to the parties. Our protocol builds on simple bit commitment and oblivious transfer protocols, using XOR commitments to prove simple relations in zero-knowledge. Besides providing very high security guarantees, our protocols are significantly simpler and more efficient than previous results, since they rely on pre-computed operations distributed by the TI.*

Keywords. Committed Oblivious Transfer, Commodity based cryptography, Unconditional Security, Universal Composability.

1. Introduction

Committed Oblivious Transfer (COT) was introduced by Crépeau, van de Graaf and Tapp in [Crépeau et al. 1995] and combines one-out-of-two oblivious transfer and bit commitment. In a nutshell, COT is a variation of Oblivious Transfer where the parties are committed to their inputs and end up committed to their outputs. Basically, this added property enables the parties to verify input correctness, since they can't change their inputs after the commitments are made. It was shown that this primitive implies efficient secure multi-party computation for dishonest majorities in the presence of a broadcast channel [Crépeau et al. 1995], while plain oblivious transfer is only able to achieve secure two-party computation without extra assumptions [Kilian 1988][Goldreich et al. 1987].

Specifically, a COT is defined as a two-party primitive where the sender's inputs are commitments to bits b_0,b_1 and the receiver's input is a commitment to a choice bit c.

In the end of the protocol, the receiver is committed to b_c but doesn't learn b_{1-c}, while the sender learns nothing about c . Security for the sender consists in the receiver learning nothing about b_{1-c} and security for the receiver consists in the sender learning nothing about c.

The original COT protocol proposed in [Crépeau et al. 1995] was proven secure in the stand-alone model, meaning that its security is only guaranteed in a very simple setting where only one copy of the protocol is executed each time. While this provides enough guarantees for simple environments, it is desirable to obtain a COT protocol with arbitrarily composable security, which guarantees that the protocol remains secure even if multiple copies of other protocols and itself are run concurrently (*e.g.*, as in the Internet). Moreover, such security guarantees allow protocols and primitives to be securely used as building blocks for more complex applications. In this paper, we adopt the Universal Composability (UC) framework introduced by Canetti *et al.* in [Canetti 2001], which is one of the main approaches for proving the security of protocols under arbitrary composition.

1.1. Committed Oblivious Transfer Applications

As mentioned before, COT was originally shown to imply unconditionally secure multiparty computation with dishonest majorities in the stand-alone model [Crépeau et al. 1995]. Nevertheless, several multiparty computation protocols based on COT have been proposed since the primitive was first introduced. We present a brief list of the main COT applications bellow:

- Universally composable unconditionally secure two-party and multi-party computation with dishonest majorities [Estren 2004] based on [Crépeau et al. 1995].
- Active-secure universally composable two-party and multi-party computation with dishonest majorities based on joint gate evaluation [Garay et al. 2004].
- Efficient active-secure universally composable two-party computation based on Yao's garbled circuits [Jarecki and Shmatikov 2007].
- Efficient active-secure universally composable two-party and multi-party computation with dishonest majorities based on Multi-sender k-out-of-n OT [Lindell et al. 2011], which is implied by COT.

In light of the broad range of applications, we remark that COT is an important building block for efficient secure two-party and multi-party computation. Both classical and recent results show that this primitive is closely related to secure computation with dishonest majorities achieving nice efficiency. However, it is important to construct COT protocols that are as efficient as possible, since the efficiency of such secure computation protocols is directly related to their building blocks.

1.2. Related Works

Building on the notions introduced in the seminal work of Crépeau, van de Graaf and Tapp [Crépeau et al. 1995], different constructions of COT with diverse security guarantees have been proposed. These constructions are mainly based on two approaches: using black-box access to primitives [Estren 2004] and building on specific computational assumptions [Garay et al. 2004, Jarecki and Shmatikov 2007, Cramer and Damgård 1997, Cachin and Camenisch 2000]. While the protocols in [Garay et al. 2004, Estren 2004,

Jarecki and Shmatikov 2007] offer universally composable security, the protocols in [Cramer and Damgård 1997, Cachin and Camenisch 2000] have only been proved in the stand-alone model.

Cramer and Damgård introduced a COT protocol based on the Q'th Residuosity assumptions with stand-alone security. Cachin and Camenisch proposed another stand-alone secure Verifiable Oblivious Transfer (VOT) protocol based on the Decisional Diffie Hellman (DDH) assumption. VOT is slightly different from COT in that the receiver learns b_C instead of a commitment to b_c, while both parties are committed to their inputs. Both protocols achieve nice efficiency but still require a number of modular exponentiations, which is a costly operation.

Estren extended the results of [Crépeau et al. 1995] to the universal composability framework in [Estren 2004]. This work is basically a UC secure version of [Crépeau et al. 1995], relying on black-box access to universally composable oblivious transfer, bit commitment with XOR and authenticated channel functionalities. It requires no extra computational assumptions and builds on coding and privacy amplification techniques. Even though the generic construction is efficient itself, its concrete efficiency depends on the specific constructions of the primitives used as its building blocks. Considering that the most efficient protocol for UC secure oblivious transfer [Peikert et al. 2008] known in current literature still requires several modular exponentiations, the resulting protocol is computationally costly.

Protocols for universally composable committed oblivious transfer based on specific computational assumptions were introduced in [Jarecki and Shmatikov 2007, Garay et al. 2004]. Both of these protocols use the common reference string model as a setup assumption. While the construction in [Jarecki and Shmatikov 2007] is based on Decisional Composite Residuosity (DCR), the result on [Garay et al. 2004] relies on multiple assumptions, including DCR, strong RSA and Decisional Diffie Hellman. The protocol in [Jarecki and Shmatikov 2007] achieves string COT, while [Garay et al. 2004] achieves bit COT. However, both protocols rely on costly operations.

1.3. Our contributions

In this paper, we introduce a universally composable committed oblivious protocol based on a Trusted Initializer (TI). The TI is an entity that predistributes values to the parties during a setup phase before the protocol is executed. In contrast to previous protocols, our construction is round-optimal, only requires simple addition operations and achieves *unconditional security* (*i.e.* the protocol does not rely on any computational assumption). The main features of our protocol are summarized as follows:

- **Unconditional security:** the protocol remains secure even against computationally unbounded adversaries.
- **Universal Composability:** the protocol is proven UC secure.
- **Computational Efficiency:** we do not require any expensive operations (*e.g.* modular exponentiation or complex underlying protocols (*e.g.* zero-knowledge proofs).
- **Round Optimal:** Our protocol requires only two rounds, which is optimal.

Trusted initializers were introduced as part of the Commodity Based Cryptography model by Beaver [Beaver 1997, Beaver 1998], which is inspired by client-server

architectures. The commodity based cryptography model makes perfect sense in the context of universal composability, since it was shown that it is necessary to rely on setup assumptions to obtain universally composable bit commitment, oblivious transfer and secure multi-party computation in general [Canetti and Fischlin 2001]. As an added advantage, the TI in our scenario can also be used to cheaply predistribute a broadcast channel, which is necessary for obtaining secure multi-party computation based on COT.

To the best of our knowledge this is the first universally composable COT protocol to be constructed on the commodity based cryptography model. Building on this model allowed us to obtain much better efficiency and security by trading off complex building blocks and assumptions for data predistribution and precomputation. In fact, we obtain the strongest possible security guarantees, namely unconditional security with universal composability. Moreover, our protocol is very simple in comparison to other protocols that require zero-knowledge proofs, verifiable encryption and universally composable building blocks.

1.4. Organization

The remainder of this paper is organized as follows. In Section 2, we establish notation and discuss preliminary notions and tools that will be used throughout the paper. In Section 3, we provide a brief discussion of the Universal Composability framework and define the ideal functionalities that will be used in this paper. In Section 4, we present the committed oblivious transfer protocol. In Section 5, we present the formal security proof for the protocol. In Section 6, we conclude with closing remarks and directions for future research.

2. Preliminaries

In this section, we establish notation and discuss the main tools and notions that we will use throughout the paper.

2.1. Notation

We will denote by $x \in_R D$ an uniformly random choice of an element x over a domain D. All logarithms are to the base 2. If X and Y are families of distributions indexed by a security parameter λ, we use $X \overset{s}{\approx} Y$ to mean the distributions X and Y are *statistically close*, *i.e.*, for all polynomials p and sufficiently large λ, we have $\sum_x |Pr[X = x] - Pr[Y = x]| < 1$. Two sequences $X_n, n \in \mathbb{N}$ and $Y_n, n \in \mathbb{N}$ of random variables are said to be *computationally indistinguishable*, denoted by $X \overset{c}{\approx} Y$, if for every non-uniform Probabilistic Polynomial Time *PPT* distinguisher D there exists a negligible function $\mathsf{negl}.(\cdot)$ such that for every $n \in \mathbb{N}$, $\mid Pr[D(X_n) = 1] - Pr[D(Y_n) = 1] \mid < \mathsf{negl}.(n)$.

Additionally we will denote by $[b]$ a commitment to a bit b and by $|b|$ the information necessary to reveal this commitment; by $\mathfrak{C}$ the co-domain set of the commitments made with a bit commitment scheme $\mathcal{C}$; by $|\mathfrak{C}|$ the set of the information necessary to open a commitment in $\mathfrak{C}$. We will use a prime (') in all the variables on the simulated environment.

2.2. Commodity Based Cryptography

In this section, we briefly discuss the Commodity Based Cryptography model introduced by Beaver [Beaver 1997, Beaver 1998]. 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 predistributed 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 predistributed to the parties. The Trusted Initializer is always assumed to be honest. Notably, this model has been used to construct commitments [Rivest 1999, Blundo et al. 2002, Hanaoka et al. 2003] and oblivious transfer [Beaver 1997].

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 predistributes information. If the parties are isolated from the TI during protocol execution it helps reduce the trust put in the TI. Moreover, data obtained from different TIs can be combined in order to thwart attacks from a single corrupted TI in more complex situations.

2.3. Bit Commitment with XOR

A bit commitment is a two-party primitive in which the sender's input is a bit b and the receiver does not have any input. There are two phases (*commit* and *unveil*), and in the end of the protocol, the receiver accepts the bit revealed bit or not. In the commit phase, the sender (or *committer*) makes some computation such that the output appears random from the receiver's point of view. It means that in this phase, the receiver learns nothing about b from the output $[b]$, which is a commitment to b. In the unveil phase, the sender sends the information necessary to decommit, $|b|$, (by making the inverse computation, or making the same computation and comparing the results) and the receiver checks if b is consistent with the commitment previously received.

For a bit commitment scheme to be considered secure, it needs to be *private, binding* and *correct*. A bit commitment scheme is private if the receiver is unable to learn anything about the sender's committed bit before the unveil phase. A bit commitment scheme is binding if the sender is unable to unveil a bit $\bar{b} \neq b$ without being caught with high probability. Finally, a bit commitment scheme is correct if it never fails for honest parties and the receiver learns the value b that the sender intended to commit to after the unveil phase.

For the sake of simplicity, lets refer to the sender as Alice and to the receiver as Bob. Our protocol requires a bit commitment scheme for which it is possible to verify XOR relationships between commitments without revealing their contents. More precisely, we want to verify the relation $b_1 = b_2 \oplus b_3$ given only the commitments $[b_1], [b_2]$ and the public bit b_3, without opening the commitments. Starting from any regular bit commitment scheme, we can use the *BCX* protocol for this purpose.

BCX was first described in [Kilian 1992] (partly attributed to Rudich and Bennett). The protocol works as follows. For Alice to commit to a bit b with a security parameter n and a bit commitment scheme $\mathcal{C}$, she chooses $b_L \in_R \{0,1\}^n$. For each bit $1 \leq i \leq n$, she sets a new string $b_R \in \{0,1\}^n$, such that $b_{iR} = b_{iL} \oplus b$, where b_{iR} is the i-th bit of b_R and b_{iL} is the i-th bit of b_L. Finally, she uses $\mathcal{C}$ to commit to all bits in b_R and in b_L. Hence, a commitment to a bit b in BCX consists in $2n$ commitments using $\mathcal{C}$. Then, the co-domain of commitments using BCX is $\mathfrak{C} = (\mathcal{C}(x)_{x\in\{0,1\}})^{2n}$, where $\mathcal{C}(x)_{x\in\{0,1\}}$ is the co-domain of commitments using $\mathcal{C}$. To unveil, Alice should unveil all $2n$ bits. Bob accepts it if $b_{iL} \oplus b_{iR} = b, \forall 1 \leq i \leq n$. So, if the information necessary to open a commitment using $\mathcal{C}$ is $|\mathcal{C}(x)_{x\in\{0,1\}}|$, the information necessary to open a commitment in BCX is $(|\mathcal{C}(x)_{x\in\{0,1\}}|)^{2n}$.

Given the commitments $[a]$,$[c]$ and the public value x, Alice proves to Bob that $a \oplus c = x$ by the following procedure. Bob chooses and sends to Alice random permutations π_a, π_b to shuffle the bit positions of a_{iL},a_{iR}, c_{iL}, and c_{iR}. Alice applies the permutations, sending to Bob $L_i = a_{iL} \oplus c_{iL}, R_i = a_{iR} \oplus c_{iR}, \forall 1 \leq i \leq n$. Bob randomly chooses and sends to Alice $V \in \{L,R\}^n$. For each i, if $V_i = L$, then Alice opens $[a_{iL}]$ and $[c_{iL}]$; else opens $[a_{iR}]$ and $[c_{iR}]$. Finally, Bob accepts the proof if $\forall 1 \leq i \leq n$, $a_{iL} \oplus c_{iL} = L_i$, if $V_i = L$; or $a_{iR} \oplus c_{iR} = R_i$ if $V_i = R$.

This simple XOR proof can be used to prove if two commitments are related to the same bit (if the proof that $[b_1] = [b_2] \oplus 0$ is accepted, then $b_1 = b_2$), or not (if the proof that $[b_1] = [b_2] \oplus 1$ is accepted, then $b_1 \neq b_2$). However, if more than one proof is generated for the same commitment, Bob may learn b before the unveil phase (*e.g.* by learning b_{iL} in the first proof and b_{iR} in the second proof).

In order to solve this issue, if a commitment $[b]$ is meant to be used for a XOR proof, Alice creates three more pairs $b^1_L \in_R \{0,1\}^n$, $b^1_R \in \{0,1\}^n$, $b^2_L \in_R \{0,1\}^n$, $b^2_R \in \{0,1\}^n$, and $b^3_L \in_R \{0,1\}^n$, $b^3_R \in \{0,1\}^n$, where each b^j_{iL}, and b^j_{iR}, $1 \leq i \leq n, 1 \leq j \leq 3$ are commitments with $b^j_{iL} \oplus b^j_{iR} = b$. Alice concatenates b^1_L, b^2_L, b^3_L and b^1_R, b^2_R, b^3_R obtaining two strings of $3n$ bits each. Bob sends a random permutation π that shuffles the $3n$ position bits. After applying the random permutation to b_L and b_R, Alice divides b_L in three n bits strings, calling them respectively b^1_L, b^2_L and b^3_L. She does the same to b_R, obtaining again b^1_R, b^2_R, b^3_R. Let's say that $b_1 = b^1_{iL} \oplus b^1_{iR}$,$b_2 = b^2_{iL} \oplus b^2_{iR}$, $b_3 = b^3_{iL} \oplus b^3_{iR}$ for all $1 \leq i \leq n$. For Alice to prove to Bob that $b_1 = b_2 = b_3 = b$, she uses the same previous proceeding to prove that $b \oplus b_1 = 0$. So, if the test is successful, b_L, b_R, b^1_L, b^1_R cannot be used any more, but Bob believes that $b_2 = b_3 = b$. At this point, Alice can discard b_L, b_R, use b^2_L, b^2_R in the XOR proof. After finishing the proof, she discards b^2_L, b^2_R, sets $b_L = b^3_L$, sets $b_R = b^3_R$, and sends the $2n$ bits of b^3_L, b^3_R to Bob.

3. Universal Composability

In this section, we present a brief discussion of the Universal Composability framework. We refer the readers to [Canetti 2001, Canetti and Fischlin 2001] for further details.

In the UC framework we consider a set of parties interacting with an Adversary $\mathcal{A}$, and a *environment* $\mathcal{Z}$. The environment is responsible for providing the inputs for the parties and $\mathcal{A}$, and receiving their outputs. All these entities are modeled as Interactive Turing Machines.

The whole point of the UC framework is that $\mathcal{Z}$ represents all activity external to the current execution of the protocol. So, in order to prove the security of a specific protocol implementation, we must define an ideal version of the protocol 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 execution of the ideal version 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 one or both parties are 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 $\mathcal{S}$ should be constructed 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. In other words, all the attacks can be possibly aimed at the real protocol are those reflected in the ideal functionality. Hence, if the ideal functionality specifies a secure primitive, any attack against a protocol that securely implements such functionality will not disrupt the functionality's security guarantees. 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.

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 $\mathcal{A}$ in the real protocol there exists an ideal adversary $\mathcal{S}$ such that a ZZZ cannot distinguish an execution of the specific protocol implementation with the parties and $\mathcal{A}$ from an execution of the ideal version with the parties and $\mathcal{S}$. Formally, we have: $\forall\, \mathcal{A},\ \exists\, \mathcal{S},\ s.t.\ \forall \mathcal{Z} : \mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, where the probability distribution is taken over all random tapes of the parties; REAL is the output of the environment $\mathcal{Z}$ when it interacts with $\mathcal{A}$ and the other parties running the real protocol, and, similarly, IDEAL is the output of the environment $\mathcal{Z}$ when it interacts with $\mathcal{S}$ and the other parties in the ideal execution of the protocol. Without loss of generality, let us define REAL and IDEAL as single bits.

In our work, we will show that the parties' random tapes are fed with the same probability distribution in the real execution and in the ideal one. Thus, we can actually show that the real execution is *perfectly* indistinguishable from the ideal simulation.

3.1. Ideal Functionalities

The Trusted Initializer functionality is defined as follows.

IDEAL FUNCTIONALITY $\mathcal{F}_{TI}$

- When first activated, choose $r_0, r_1, d \in_R \{0,1\}$, compute the commitments $[r_0], [r_1], [d], [r_d]$ (where $r_d \in \{r_0, r_1\}$), set $|r_0|, |r_1|, |d|, |r_d|$ as the information to open the respectively commitments, and distribute $(r_0, r_1, |r_0|, |r_1|, [d], [r_d])$ to Alice and $(d, r_d, |d|, |r_d|, [r_0], [r_1])$ to Bob .

The following COT ideal functionality $\mathcal{F}_{COT}$ is based on functionalities presented in [Estren 2004, Jarecki and Shmatikov 2007]. Each commitment has a unique identifier *cid* and each $\mathcal{F}_{COT}$ instance has a unique identifier *sid*. The parties P_R, P_S refer to Sender and Receiver, respectively, and P_i may refer to any of them. $\mathcal{S}$ is the ideal world adversary.

Ideal Functionality $\mathcal{F}_{COT}$

- Upon receiving a (COMMIT, P_i,*sid*, *cid*, b) message from P_i:
  - If *cid* has not been used for any previous commitment, then store b and broadcast a (RECEIPT, P_i,*sid*, *cid*, $[b]$), where $[b]$ is the commitment of bit b with identifier *cid*.
  - Otherwise, do nothing.
- Upon receiving $msg =$ (STARTCOT, P_S, P_R, *sid*, $cid_0, cid_1, cid_n, cid_c$) from P_R:
  - If cid_0 and cid_1 refer to existing valid unopened commitments by P_S to the respective bits b_0 and b_1, cid_c refers to an existing unopened commitment by P_R to bit c, and cid_n does not refer to any previous commitment, then record msg and forward it to P_S.
  - If any of the conditions defined in the immediately previous item fails, broadcast message (COTFAILED, P_S, P_R,*sid*).
- Upon receiving (COMPLETECOT, P_S, P_R, *sid*, $cid_0, cid_1, cid_n, cid_c$) from P_S:
  - $\mathcal{F}_{COT}$ verifies that a message (STARTCOT, P_S, P_R, *sid*, $cid_0, cid_1, cid_n, cid_c$) has been recorded. In such a case, $\mathcal{F}_{COT}$ generates a commitment to b_c under cid_n and sends a message (TRANSFERCOT, P_S, P_R, *sid*, $cid_0, cid_1, cid_n, cid_c, b_c$).
  - If no such record exists, $\mathcal{F}_{COT}$ sends message (COTFAILED, P_S, P_R,*sid*).

4. The Protocol

In this section we describe our COT protocol. Our protocol is constructed in the commodity based model, where a Trusted Initializer predistributes a number of correlated values to Alice and Bob, allowing them to perform COT. Alice inputs commitments $[b_0], [b_1]$ to bits b_0, b_1 for which she knows the opening information $|b_0|, |b_1|$, whereas Bob knows $[b_0], [b_1]$. Likewise, Bob inputs a commitment $[c]$ to a bit c for which he knows the opening information $|c|$, whereas Alice knows $[c]$. In the end of the protocol, Alice learns nothing, while Bob receives b_c and the opening information $|b_c|$ but learns nothing about b_{1-c}.

Intuitively, this protocol builds on the techniques of [Crépeau et al. 1995], combining a plain oblivious transfer protocol and a bit commitment scheme with XOR.

We obtain the commitment scheme with XOR from any regular bit commitment (*e.g.* [Rivest 1999]) through the BCX construction [Kilian 1992]. The basic idea is to have both parties run the OT protocol with committed inputs instead of plain bits. The OT protocol is slightly modified so that, in the end of the protocol, the receiver learns both the commitment corresponding to b_c and the necessary opening information (*i.e.* learning b_c itself). The receiver then uses the XOR property of the commitment scheme to check that the transferred commitment is indeed one of Alice's initial commitments (using the procedures described in Section 2.3).

The basic oblivious transfer protocol used in our construction is predistributed by the TI as Alice's values $r_0, r_1 \in_R \{0,1\}$ and Bob's values $d \in_R \{0,1\}$, $r_d \in \{r_0, r_1\}$. In order to run the OT protocol, Bob computes $e = c \oplus d$ and sends it to Alice in Step 1. Alice computes $f_0 = b_0 \oplus r_e$, $f_1 = b_1 \oplus r_{1-e}$ and sends these values to Bob in Step 2. Finally, Bob recovers bit b_c by computing $b_c = f_c \oplus r_d$ in Step 3. Notice that $f_c = b_c \oplus r_d$ and $f_{1-c} = b_{1-c} \oplus r_{1-d}$. Hence, Bob can't recover b_{1-c} because he does not know r_{1-d}, while Alice can't recover c because she does not know d.

	Alice	Bob
TI	$r_0, r_1 \in_R \{0,1\}$ $[d], [r_d], \lvert r_0\rvert, \lvert r_1\rvert$	$d \in_R \{0,1\}$ $r_d \in \{r_0, r_1\}$ $[r_0], [r_1], \lvert d\rvert, \lvert r_d\rvert$
Data sent by TI	$\mu_A := (r_0, r_1, \lvert r_0\rvert,$ $\lvert r_1\rvert, [d], [r_d])$	$\mu_B := (d, r_d, \lvert d\rvert,$ $\lvert r_d\rvert, [r_0], [r_1])$
Input	$b_0, b_1 \in \{0,1\}$ $\lvert b_0\rvert, \lvert b_1\rvert, [c]$	$c \in \{0,1\}$ $\lvert c\rvert, [b_0], [b_1]$
Step 1		$e = c \oplus d$ Send $\mu_1 := (e)$ to Alice
Step 2	If $e \notin \{0,1\}$, then abort Otherwise, do: $f_0 := b_0 \oplus r_e$ $f_1 := b_1 \oplus r_{1-e}$ Compute: $[f_0], \lvert f_0\rvert$ and $[f_1], \lvert f_1\rvert$ Send $\mu_2 := ([f_0], [f_1],$ $\lvert f_0\rvert, \lvert f_1\rvert, f_0, f_1)$ to Bob	
Step 3		Check if $f_0 \neq [b_0] \oplus [r_e]$ or $f_1 \neq [b_1] \oplus [r_{1-e}]$, then abort Otherwise, open f_0, f_1, Set $b_c = f_c \oplus r_d$ and if $(f_0, f_1) = (0,0)$ then $[b_c] := [r_d]$ if $(f_0, f_1) = (1,1)$ then $[b_c] := \neg[r_d]$ if $(f_0, f_1) = (0,1)$ then $[b_c] := [c] + [r_d]$ if $(f_0, f_1) = (1,0)$ then $[b_c] := \neg[c] + [r_d]$
Output	$\perp$	$v := [b_c], b_c$

The COT protocol combines the aforementioned OT protocol with BCX commitments by having the TI predistribute commitments $[d], [r_d]$ to Alice and commitments $[r_0], [r_1]$ to Bob, besides the regular OT predistributed values. Moreover, as this is a COT

protocol, Alice is required to send Bob the commitments $[b_0], [b_1]$ and Bob is required to send Alice the commitment $[c]$. The OT protocol is run with its normal inputs, except that Alice also sends commitments $[f_0], [f_1]$ and the respective opening information in Step 2. Upon receiving those commitments Bob uses the XOR property to check that they correspond to $f_0 = [b_0] \oplus [r_e]$ and $f_1 = [b_1] \oplus [r_{1-e}]$. If at least one of these relations hold, he continues by determining the commitment $[b_c]$ by checking the relations over (f_0, f_1) specified in Step 3 of the COT protocol.

Notice that Bob can open $[b_c]$, since he has the information to open $[c]$ and $[r_d]$. In Step 3, Bob checks whether Alice set the masks f_0, f_1 appropriately (i.e. f_0, f_1 are not random numbers), in order to verify that he correctly received one of Alice's input bits. However, we have to verify that $f_0 = b_0 \oplus r_e$ and $f_1 = b_1 \oplus r_{1-e}$ without revealing any of the bits b_0, b_1, r_0, r_1. This is the reason why we need a bit commitment with XOR, meaning that it has the property of evaluating XOR between commitments without having to open them. If the commitment scheme is homomorphic, we don't require this XOR property, since a sum of commitments is also a commitment (*i.e.*, $[b_0] + [r_e] = [b_0 + r_e]$) and Alice cannot deviate from the protocol. A extended version of this protocol using a homomorphic bit commitment scheme will be included in the full version.

4.1. Correctness

First we analyze correctness in the case that both parties are honest.

Alice learns nothing about c. Bob sends information about c only in Step 1. In this step, Bob sends $e = c \oplus d$, and provided that Alice cannot extract d from the commitments $[d]$ and $[r_d]$, $c \oplus d$ behaves as a one time pad encryption of c, since Alice does not know d. Hence, Alice can only guess the value of c.

Bob learns nothing about $b_{\bar{c}}$. Notice that if $c = 0$, then $e = d$ and $f_1 = b_1 \oplus r_{1-d}$. Conversely, if $c = 1$ then $e = 1 - d$ and $f_0 = b_0 \oplus r_{1-d}$. Also notice that Alice computes $f_{1-c} = b_{1-c} + r_{1-d}$ and Bob doesn't know r_{1-d}. Hence, he is not able to compute b_{1-c} from f_{1-c}.

5. Universally Composable Security Analysis

In this section we construct the simulators and prove that our protocol is universally composable.

Remember that the players in a real world protocol execution interact with the environment $\mathcal{Z}$, with the TI, and with an adversary $\mathcal{A}$, which will be called the real-life adversary. On the other hand, the players in a execution of the ideal process interact with the environment $\mathcal{Z}$ and TI as well, and with an ideal adversary $\mathcal{S}$ (the simulator).

In order to prove the UC security, we have to construct an ideal adversary $\mathcal{S}$ such that the environment cannot guess whether it is interacting with $\mathcal{A}$ in the real-life execution or with $\mathcal{S}$ in the ideal simulation with more than a negligible probability. So, we will construct $\mathcal{S}$ by showing the actions that she takes in the ideal process for each situation: (Alice corrupted $\wedge$ Bob honest) $\vee$ (Alice honest $\wedge$ Bob corrupted) $\vee$ (Alice corrupted $\wedge$ Bob corrupted) $\vee$ (Alice honest $\wedge$ Bob honest).

5.1. Corrupted Alice and Honest Bob

Simulation. In this case only Alice is corrupted, while Bob acts as an honest party. The simulator $\mathcal{S}$ runs an internal copy of the real-life adversary $\mathcal{A}$ called A' and every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is forwarded to A' without any alteration. The interactions between A' and $\mathcal{S}$ correspond to those of Alice in the real protocol execution with $\mathcal{Z}$, TI, and Bob. $\mathcal{S}$ operates as follows:

1. **Setup Phase:**
   - Simulating $\mathcal{F}_{TI}$: $\mathcal{S}$ chooses $r'_0, r'_1 \in_R \{0,1\}$, $[d'], [r'_{d'}] \in_R \mathfrak{C}$, $|r'_0|, |r'_1| \in_R |\mathfrak{C}|$, and sends $\mu'_A = (r'_0, r'_1, |r'_0|, |r'_1|, [d'], [r'_{d'}])$ to A' when it requests data from $\mathcal{F}_{TI}$.
2. **Protocol**:
   (a) Simulating message μ'_1: $\mathcal{S}$ chooses random bits $d, c \in_R \{0,1\}$ and $r_d \in_R \{r'_0, r'_1\}$, computes $e = c \oplus d$ and sends $\mu'_1 = e$ to corrupted Alice.
   (b) Extracting b_0, b_1: Upon receiving message $\mu_2 := ([f'_0], [f'_1], |f'_0|, |f'_1|, f'_0, f'_1)$ from corrupted Alice, $\mathcal{S}$ computes $b'_0 = f'_0 \oplus r_e$ and $b'_1 = f'_1 \oplus r_{1-e}$ (notice that $\mathcal{S}$ is able to do so, since it knows r_0, r_1). $\mathcal{S}$ sends the messages (COMMIT, P_S,*sid*, cid_0, b'_0), and (COMMIT, P_S,*sid*, cid_1, b'_1) to $\mathcal{F}_{COT}$.
   (c) Finishing the protocol: Upon receiving a message (STARTCOT, P_S, P_R, $sid, cid_0, cid_1, cid_n, cid_c$) from $\mathcal{F}_{COT}$, $\mathcal{S}$ sends a message (COMPLETECOT, P_S, P_R, $sid, cid_0, cid_1, cid_n, cid_c$) to $\mathcal{F}_{COT}$.

Indistinguishability. In order to prove that $\forall\mathcal{A}\exists\mathcal{S}\forall\mathcal{Z} : \text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \equiv \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, we have to show that the view of the environment $\mathcal{Z}$ in the execution of the real-life protocol in the presence of the real-life adversary $\mathcal{A}$ is indistinguishable from the view of $\mathcal{Z}$ in the execution of the ideal protocol in the presence of the ideal adversary $\mathcal{S}$.

Notice that the values $r'_0, r'_1 \in_R \{0,1\}$, $[d'], [r'_{d'}] \in_R \mathfrak{C}$, $|r'_0|, |r'_1| \in_R |\mathfrak{C}|$ simulated by $\mathcal{S}$ in the setup phase have the same distribution as in the real protocol. Hence the pre-distributed values are perfectly indistinguishable from values in the real world execution.

Apart from that, the only difference in the simulated protocol is that $\mathcal{S}$ chooses a random value for c. However, since d is also chosen randomly, $e = c \oplus d$ acts as an one time pad encryption of c. Thus, there's no difference in the distribution of the values sent to corrupted Alice in the simulation.

Notice that $\mathcal{S}$ obtains corrupted Alice's inputs b'_0, b'_1 since it knows r'_0, r'_1, which $\mathcal{S}$ chose itself in the setup phase.

5.2. Honest Alice and Corrupted Bob

Simulation.

As in the previous case, $\mathcal{S}$ runs an internal copy of the real-life adversary $\mathcal{A}$ called B', and every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is forwarded to B' without any alteration. The interactions between B' ansd$\mathcal{S}$ are those of Bob in the real protocol execution with $\mathcal{Z}$, the TI and Alice. $\mathcal{S}$ works as follows:

1. **Setup Phase:**

- Simulating $\mathcal{F}_{TI}$: $\mathcal{S}$ chooses $d', r'_{d'} \in_R \{0,1\}$, $|d'|, |r'_{d'}| \in_R |\mathfrak{C}|$, $[r'_0], [r'_1] \in_R \mathfrak{C}$, and sends $\mu'_B = (d', r'_{d'}, |d'|, |r'_{d'}|, [r'_0], [r'_1])$ to B'.

2. **Protocol**:
   - Extracting c: Upon receiving message $\mu'_1 = (e)$, $\mathcal{S}$ computes $c = e \oplus d'$ (notice that this is possible since $\mathcal{S}$ knows d'). $\mathcal{S}$ sends message (COMMIT, P_i,*sid*, *cid*$_c$, c) to $\mathcal{F}_{COT}$.
   - Obtaining b_c: $\mathcal{S}$ waits for two messages messages (RECEIPT, P_S,*sid*, *cid*$_0$, $[b'_0]$) and (RECEIPT, P_S,*sid*, *cid*$_1$, $[b'_1]$) from $\mathcal{F}_{COT}$ and then sends a message (STARTCOT, P_S, P_R, *sid*, *cid*$_0$, *cid*$_1$, *cid*$_n$, *cid*$_c$) to $\mathcal{F}_{COT}$.
   - Simulating μ'_2 and Finishing the protocol: Upon receiving a message (TRANSFERCOT, P_S, P_R, *sid*, *cid*$_0$, *cid*$_1$, *cid*$_n$, *cid*$_c$, b'_c), $\mathcal{S}$ selects random values $r'_0, r'_1 \in_R \{0,1\}$, $[d'], [r'_{d'}] \in_R \mathfrak{C}$, $|r'_0|, |r'_1| \in_R |\mathfrak{C}|$ and a random message b'_{1-c}. It computes $f'_0, f'_1, [f'_0], [f'_1], |f'_0|, |f'_1|$ as in the real protocol using messages b'_c, b'_{1-c} (where b'_c was obtained from $\mathcal{F}_{COT}$.

Indistinguishability. As in the previous case, all the predistributed values $d', r'_{d'} \in_R \{0,1\}$, $|d'|, |r'_{d'}| \in_R |\mathfrak{C}|$, $[r'_0], [r'_1] \in_R \mathfrak{C}$ simulated by $\mathcal{S}$ are distributed exactly as in the real world protocol. Hence, the only difference lies in message b'_{1-c}, that is randomly selected.

It is easy to notice that $\mathcal{S}$ can obtain c by computing $c = e \oplus d$, since it generated d and knows its value.

However, since the only information that corrupted Bob obtains about b'_{1-c} is $f_{1-c}, |f_{1-c}|, [f_{1-c}]$, corrupted Bob cannot distinguish b'_{1-c} in the simulation from the message in the real execution, since $f_{1-c} = b_{1-c} \oplus r_{1-d}$ and corrupted Bob doesn't know r_{1-d}, which is randomly selected.

Notice that the distribution of all values exchanged between $\mathcal{S}$ corrupted Bob and the environment is exactly the same as in the real world execution. Thus, the simulation is perfectly indistinguishable from the real world protocol execution.

5.3. Both parties are honest or corrupted

When both parties are corrupted or honest, $\mathcal{S}$ simply forwards all messages between parties and from parties to $\mathcal{F}_{COT}$.

6. Conclusion

We introduce the first unconditionally secure and universally composable committed oblivious transfer protocol in the commodity based cryptography model. Our protocol achieves the highest security guarantee known in current literature while conserving communication efficiency, since it is round-optimal. Moreover, being based on predistributed values and precomputation by a Trusted Initializer, our protocol is extremely efficient, not requiring any costly operations such as modular exponentiations. We remark that this COT protocol can also be used as a verifiable oblivious transfer protocol with minor modifications. As a future work, we suggest to obtain a modified version of this protocol based on homomorphic commitments, which could achieve even better efficiency. Another interesting related line of work is applying the commodity based cryptography model to perform secure two-party computation on arithmetic circuits (*i.e.* in $\mathbb{Z}_p$) without relying on COT.

Acknowledgement

The work of the third author was partially supported by a FAPEMIG grant, number APQ-02719-10.

References

Beaver, D. (1997). Commodity-based cryptography (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 446–455, New York, NY, USA. ACM.

Beaver, D. (1998). Server-assisted cryptography. In *Proceedings of the 1998 workshop on New security paradigms*, NSPW '98, pages 92–106, New York, NY, USA. ACM.

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

Cachin, C. and Camenisch, J. (2000). Optimistic fair secure computation. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 93–111, London, UK, UK. Springer-Verlag.

Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS '01, pages 136–, Washington, DC, USA. IEEE Computer Society.

Canetti, R. and Fischlin, M. (2001). Universally composable commitments. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 19–40, London, UK. Springer-Verlag.

Cramer, R. and Damgård, I. (1997). Linear zero-knowledge - a note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 436–445, New York, NY, USA. ACM.

Crépeau, C., van de Graaf, J., and Tapp, A. (1995). Committed oblivious transfer and private multi-party computation. In Coppersmith, D., editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123. Springer.

Estren, G. (2004). Universally composable committed oblivious transfer and multi-party computation assuming only basic black-box primitives. Master's thesis, McGill University.

Garay, J. A., Mackenzie, P., and Yang, K. (2004). Efficient and universally composable committed oblivious transfer and applications. In *Proceedings of the First Theory of Cryptography Conference (2004*, pages 297–316. Springer-Verlag.

Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA. ACM.

Hanaoka, G., Imai, H., M′uller-Quade, J., Nascimento, A., and Otsuka, A. (2003). Unconditionally secure homomorphic pre-distributed bit commitment. *ICS 2003*.

Jarecki, S. and Shmatikov, V. (2007). Efficient two-party secure computation on committed inputs. In *Proceedings of the 26th annual international conference on Advances in Cryptology*, EUROCRYPT '07, pages 97–114, Berlin, Heidelberg. Springer-Verlag.

Kilian, J. (1988). Founding crytpography on oblivious transfer. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, New York, NY, USA. ACM.

Kilian, J. (1992). A note on efficient zero-knowledge proofs and arguments. *Proceeding STOC '92 Proceedings of the twenty-fourth annual ACM symposium on Theory of Computing*, pages 723 – 732.

Lindell, Y., Oxman, E., and Pinkas, B. (2011). The ips compiler: Optimizations, variants and concrete efficiency. *Advances in Cryptology–CRYPTO 2011*, pages 259–276.

Peikert, C., Vaikuntanathan, V., and Waters, B. (2008). A framework for efficient and composable oblivious transfer. In Wagner, D., editor, *Advances in Cryptology ? CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer Berlin / Heidelberg.

Rivest, R. L. (1999). 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.

