Essentially Optimal Universally Composable Oblivious Transfer

Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi

BRICS, Department of Computer Science, Aarhus Universitet,
Åbogade 34, 8200 Århus, Denmark
{ivan,buus,orlandi}@daimi.au.dk

Abstract Oblivious transfer is one of the most important cryptographic primitives, both for theoretical and practical reasons and several protocols were proposed during the years. We provide the first oblivious transfer protocol which is simultaneously optimal on the following list of parameters: *Security:* it has universal composition. *Trust in setup assumptions:* only one of the parties needs to trust the setup (and some setup *is* needed for UC security). *Trust in computational assumptions:* only one of the parties needs to trust a computational assumption. *Round complexity:* it uses only two rounds. *Communication complexity:* it communicates $\mathcal{O}(1)$ group elements to transfer one out of two group elements. The Big-O notation hides 32, meaning that the communication is probably not optimal, but is essentially optimal in that the overhead is at least constant. Our construction is based on pairings, and we assume the presence of a key registration authority.

Key words: Oblivious Transfer, Universally Composable Security.

1 Introduction

An oblivious transfer (OT) involves two parties, a sender and a receiver. The sender has two secret messages. The receiver selects to retrieve one of them, without disclosing which one. At the same time the receiver is not allowed to learn more than one secret. Oblivious transfer was first introduced by Wiesner [Wie83] in the late seventies under the name of *conjugate coding*. However, the importance of this primitive in the cryptographic field was first pointed out by Rabin in [Rab81].

OT is the base for many secure multiparty computation (SMC) protocols [Yao86, GMW87], where several instances of OT are run at the same time. However, many of the proposed OT protocols do not give any guarantee about the security under composition. We present here the first protocol that is secure in the universally composable [Can01] model, using just two rounds of communication and having only a constant overhead ($\mathcal{O}(k)$ bits are communicated to do an OT of k bits).

In addition we do not need to assume a common reference string. Instead the receiver R once and for all has to register a public key with a key registration authority KR and prove to KR that it knows the corresponding secret key. After this any sender S can retrieve the public key and perform an OT to R — in particular, S needs to have no public key or trust any common reference string, giving our construction the same flavor as a PKI for public-key encryption, where also just R registers a public key, after which all S can transfer messages securely to R. The public-key flavor of our protocol makes ideal for asymmetric settings where, e.g., one party is a server, which can afford the time and cost of registering a public key for the given application. Clients then need only retrieve the public key of the server to perform UC OT with the server.

As a final feature our protocol is perfectly secure for the sender. Our protocol can therefore be viewed as optimal in four respects:

1. It is secure under general composition.
2. It uses two rounds. Clearly there exists no one-round OT protocol.
3. Only one party has to trust the setup — S has to trust that KR checks that R knows its secret key. Since OT is impossible in the plain UC model, some setup must be trusted, and having only one party do so is optimal.

4. Only one party has to trust a computational assumption. In particular, S has perfect security. No OT for the classical model can have perfect security for both parties.

The communication complexity is probably not optimal. Under the decisional linear (DLIN) assumption we send 32 group elements to transfer one out of two group elements. This gives a constant overhead, but is probably far from optimal. This seems, so far, to be the unfortunate price to pay for the other fully optimal properties.

Our OT protocol is primarily based on homomorphic encryption in pairing friendly groups, which we use to give a new instantiation of the notion of mixed commitments from [DN02]. This instantiation is constructed to work well with the efficient non-interactive zero-knowledge (NIZK) proofs by Groth, Ostrovsky and Sahai [GOS06, GS08], that are in turn based on paring-based cryptography. We put ourselves in the hybrid UC model, where all parties have access to secure and authenticated channels, and to a key registration authority (KR). This model was presented and motivated in [BCNP04].

Related Work. Examples of two-round OT can be found in [NP01, AIR01, Kal05]. However, none of these protocols achieve UC security. If we consider UC security, OT protocols are known, but they require more rounds of interaction [Gar04, JS07] or other parties helping the computation [Fis06].

As a witness that a secure and efficient OT is of primary importance, several attempts were made in the last years. Lindell [Lin08] has a very general construction that achieve full simulation, based on the existence of homomorphic encryption solely. Camenisch, Neven and shelat [CNS07] built a protocol for adaptive k-out-of-n OT, providing full simulation with specific number-theoretic assumptions. Upon this work Green and Hohenberger [GH07] built another adaptive OT, that requires weaker assumptions.

Independently from our work Peikert, Vaikuntanathan and Waters [PVW07], presented a two round UC OT protocol. However, their protocol works in the common reference string, and uses different computational assumptions, therefore these two works can be seen as complementary.

2 Main Ideas

In this section we are going to give the main ideas, leaving all the details to the rest of the paper. We first present an attack that motivates the need of a composable OT, then we sketch the protocol.

We have two players called the sender (S) and the receiver (R). The sender has two secrets x_0, x_1, while the receiver has a selection bit b. At the end of the protocol R gets x_b while S gets nothing.

2.1 Insecurity of OT Composition

We can describe a round optimal OT (i.e., 2 round OT) in the following way:

Choose: R computes a message $c = \mathrm{Choose}(b)$, and sends c to S.
Transfer: S computes a message $t = \mathrm{Transfer}(c, x_0, x_1)$ and sends it to R.
Retrieve: R retrieves $x_b = \mathrm{Retrieve}(t)$. [1]

The security of such a protocol is usually stated like:

Receiver's privacy: the output of the Choose phase, c, does not reveal any information about b to S.
Sender's privacy: the output of the Transfer phase, t, does not reveal anything about x_{1-b} to R.

[1] Note that this is the only possible order of the messages. If we build a protocol where S sends the first message and then R computes x_b from this message, then clearly R can choose to learn both x_0 and x_1.

This kind of security definition works in the case of a stand-alone OT execution, but fails dramatically in the case of even a sequential composition, and even for very generic reasons. We consider the following composed protocol to illustrate it: R and S run a first OT protocol. R inputs b, S inputs x_0, x_1, and R gets x_b. Then R and S run a second OT protocol. R inputs b', S inputs x'_0, x'_1, and R gets $x'_{b'}$. Then R sends $x'_{b'}$ to S. Now instantiate this protocol with a two-message OT secure according to the previous definition:

1. (a) R computes $c = \text{Choose}(b)$, and sends c to S.
   (b) S computes $t = \text{Transfer}(c, x_0, x_1)$ and sends it to R.
   (c) R retrieves $x_b = \text{Retrieve}(t)$.
2. (a) R computes $c' = \text{Choose}(b')$, and sends c' to S.
   (b) S computes $t' = \text{Transfer}(c', x'_0, x'_1)$ and sends it to R.
   (c) R retrieves $x'_{b'} = \text{Retrieve}(t')$.
3. R sends $x'_{b'}$ to S.

A cheating S could use, in the second Transfer phase, the first Choose message, i.e., it could compute $t' = \text{Transfer}(c, x'_0, x'_1)$. Therefore R will retrieve x'_b instead of $x'_{b'}$, without noticing it, and in the 3rd step, it will send x'_b to S, clearly revealing information about b, which an ideal implementation would not.

Note that, despite the fact that the protocol presented is an ad-hoc constructed counterexample, this vulnerability is actually quite important and has many consequences: when parties run more OTs instances, the receiver cannot be sure that the Transfer messages contain his choice. A protocol that is not secure against this attack is the one in [AIR01].

Intuitively this problem arises from the fact that the security of the sender and the security of the receiver are analyzed separately, and therefore there is no “link” between the Choose phase and the Transfer phase[2]. Another common definition for the security of OT protocols is the half-simulation, as in [NP05]. In this scenario we usually require strong (simulation) security against the receiver, but just stand-alone privacy against the sender. Note that this would not protect against the attack sketched above. This relaxation is usually justified by saying that the sender is commonly a server or a service provider, and therefore it can be controlled better or more than the receiver, who represents any user. As the above example shows, this motivation assumes that the server chooses not to learn information, which it could in fact learn by deviating only so slightly from the implementation.[3] Under such an assumption (essentially that the server is at most passively corrupted) things become much simpler. Here we want active security for both parties.

2.2 Our Protocol

We are going to present the main intuition behind our protocol in 5 steps.

Step 1: OT based on Homomorphic Encryption. Assume to have available an additively homomorphic cryptosystem, i.e., a cryptosystem that satisfies the following: $D(E(x)E(y)) = x + y$, where E, D represent the

[2] A way to fix this problem, as some OT protocols do, is to change the structure of the protocol, allowing Choose to output also a piece of trapdoor information k that will be later used during the retrieve phase. In this case the protocol will be of the form: $(c, k) = \text{Choose}(b); t = \text{Transfer}(c, x_0, x_1); x_b = \text{Retrieve}(t, k)$. We prefer, instead of fixing just this problem, to develop our protocol in the UC framework, for it provides us stronger guarantees. In particular, UC security protects against ill effects of composition as that described above, while still allowing us to analyze the protocol in isolation.

[3] Also this motivation is not so strong given that in several applications the role of the sender and the receiver can be swapped. Moreover, in some applications like authentication, it is the server that plays the role of the receiver, while the user plays the role of the sender. This could in principle be handled by using that OT is symmetric: an OT from S to R can be turned into an OT from R to S without further assumptions. This transformation however, adds another round of communication, and it always produces a one-bit OT. For applications where two-round OT or string-OT is needed, “turning the OT around” is therefore not a practical solution.

encryption and the decryption functions[4]. Then the following is a simple OT construction if the parties are semi-honest:

Choose: The receiver encrypts $c_1 = E(b)$ and sends it to the sender.
Transfer: The sender computes $c_0 = E(1)c_1^{-1} = E(1-b)$ and $d = c_0^{x_0}c_1^{x_1}$ and sends d it to the receiver.
Retrieve: The receiver decrypts $x_b = D(d)$.

The idea is that the receiver let (c_0, c_1) be an encryption of the vector $(1,0)$ if he wants to get the first secret or $(0,1)$ if he wants the second one. The sender computes, exploiting the homomorphic property: $c_0^{x_0}c_1^{x_1} = E((1-b)x_0 + bx_1) = E(x_b)$.

Step 2: Managing a Malicious Receiver. The protocol is intuitively correct and private if both parties follow the protocol. However, in the case of malicious adversaries there are many evident security flaws. First of all, if the receiver is not honest, he could send an encryption of $b \notin \{0,1\}$. If for instance he sends an encryption of $b = 2$, at the end of the protocol he will get $2x_1 - x_0$, which leaks both x_0 and x_1 if they are bits. Therefore the receiver has to prove that the message is well formed, by using a NIZK proof.

Step 3: Managing a Malicious Sender. A sender, even behaving maliciously, cannot break the privacy of the receiver without breaking the security of the underlying encryption scheme. In fact, he just sees a ciphertext. The receiver, however, has no way to check if the sender is inputting a "fresh" ciphertext, obtained through the expected computation or just a chosen ciphertext that came from somewhere else. The actual problem is that we do not check if the sender knows what he is inputting or not, and so the output may not even depend at all on the Choose message. This kind of problem does not allow the sender to learn more than what he is supposed to in a single execution of the protocol. But, as described in Section 2.1, if more than one instance of the protocol is run, the loss of security is dramatic. Therefore we have to ask also the sender to prove that the message is well formed, using a NIZK proof.

Step 4: Achieving UC. To achieve UC we need to be able to simulate the view of the parties in the real protocol, by having access just to the ideal functionality. It is well known that it is impossible to achieve UC OT in the plain model [CF01], and therefore we will assume to have access to a KR authority [BCNP04]. The protocol then consists of two phases. In the registration phase the receiver once and for all registers an encryption key ek at the KR and proves that he knows the corresponding decryption key dk. In the communication phase the parties then perform the actual OTs using the encryption scheme $E = E_{ek}$. In the simulation it is the simulator who simulates the KR authority, which allows it to extract the decryption key dk known by the receiver. This allows the simulator to compute from c_1 the choice bit used by a corrupted receiver.

In the case of a malicious sender things are more difficult. In fact, to be able to simulate, we need to extract both x_0, x_1 from the message d. But a well formed d contains no information at all about one of the two secrets. To deal with this we need the receiver to register another key ck with the KR, where ck is a public key for a commitment scheme. Then the sender commits to x_0 and x_1 under ck and gives a NIZK proof that the commitments indeed contain messages used to compute the reply d from c_0, c_1. By using a perfectly hiding commitment scheme, the receiver will not be able to learn anything from these commitments in the real protocol. In the simulation we will, however, let the simulator cheat and use an extractable commitment scheme, which allows it to extract x_0 and x_1 from the commitments.

Step 5: The Final Protocol. The UC functionality we want to implement is:

Choose: R inputs $(\texttt{choose}, otid, b)$ to the ideal functionality, where $otid$ must be a fresh OT identifier.

[4] Note that repeating this operation we can also achieve that $D(E(x)^a) = ax$.

Transfer: The ideal functionality outputs $(\texttt{chosen}, otid)$ to S. S can then any number of times input a message of the form $(\texttt{transfer}, otid, x_0, x_1)$ into the ideal functionality.

Retrieve: Each time the ideal functionality outputs $(\texttt{retrieve}, S, otid, x_b)$ to R.

This ideal functionality slightly generalizes the standard one in that S can transfer several times using the same choose message from R. This e.g. allows OT of longer strings efficiently. Collecting the above five steps we get the following protocol:

Key-Registration: R registers an encryption key ek and a commitment key ck and proves that it knows the decryption key dk (corresponding to ek), and that ck is a key for a perfect hiding commitment scheme. If so, the public keys ek and ck are given to S.

Choose: R gets his selection bit b. He encrypts $c_1 = E_{ek}(b)$. He computes a NIZK proof π_{choose} that c_1 contains either 0 or 1, and sends (c_1, π_{choose}) to S.

Transfer: S gets his two inputs x_0, x_1. He gets c_1, π_{choose} and he aborts if π_{choose} is invalid. If not, he computes $c_0 = E(1)c_1^{-1}$ and $d = c_0^{x_0} c_1^{x_1}$. Then he computes $C_0 = \mathrm{Comm}_{ck}(x_0), C_1 = \mathrm{Comm}_{ck}(x_1)$. He finally computes a NIZK $\pi_{transfer}$ that proves that d is computed correctly from c_1 and the values inside the commitments C_0, C_1. He sends everything to R.

Retrieve: R gets $d, C_0, C_1, \pi_{transfer}$ and he aborts if the check on $\pi_{transfer}$ fails. If not, he decrypts $D_{dk}(d) = x_b$ and he outputs it.

There are a number of technical issues which we solved to make the above approach work: The "encryption scheme" used is in fact a mixed commitment scheme, that is built on top of Boneh, Boyen and Shacham [BBS04] cryptosystem. Next, this mixed commitment scheme has no efficient decryption, i.e., the receiver gets an element of the form g^{x_b}, with x_b the message that the sender inputs in the OT. We will therefore start with the description of a bit OT protocol, where this is clearly not an issue. If we want to transmit more data, we can think of our protocol as a random OT, where the sender picks x_0 and x_1 at random and the receiver receives $K_0 = g^{x_0}$ or $K_1 = g^{x_1}$. Even though the sender cannot choose K_0 and K_1 as it desires, he can compute them on his side. Together with the second message the sender then sends $E_{K_0}(m_0)$ and $E_{K_1}(m_1)$, where (m_0, m_1) are the actual messages of the OT and $E_{K_b}(m_b)$ is an encryption of m_b under the key K_b. In this way we avoid the discrete logarithm problem. Another issue is that the NIZK proofs from [GOS06, GS08] work in the CRS model, while we prefer to put ourselves in the KR model to have just one party trust the setup. We deal with this by noting that the NIZK proofs can be instantiated with either perfect soundness or perfect ZK, depending on how the CRS is created. We let R register two CRSs, one of each flavor as part of his public key. When R proves, he uses the scheme with perfect soundness. When S proves, he uses the scheme with perfect zero-knowledge.

3 Preliminaries

3.1 Pairing-Based Cryptography

In the last years pairing-based cryptography gained more and more interest. Since its introduction in [Jou00][5], pairings were used in several applications and allowed to achieve strong goals, like IBE [BF01].

In pairing-based cryptography we can define bilinear maps between groups of points on elliptic curves as follows:

[5] In fact, pairings were used even before in cryptography, in order to break the discrete logarithm problem (the MOV attack [MOV93]).

Definition 1. *Let $\mathbb{G}, \mathbb{G}_1$, be two multiplicative cyclic groups of finite order n, and let g be a generator for $\mathbb{G}$. Then we say that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ is a bilinear map if:*

Bilinear: *e is bilinear, i.e., for all $x, y \in \mathbb{G}$, $a, b \in \mathbb{Z}$ we have that $e(x^a, y^b) = e(x, y)^{ab}$.*
Non-Degenerate: *For all $x \in \mathbb{G}$, $x \neq 1$, $e(x, x)$ generates $\mathbb{G}_1$.*
Computable: *For all $x, y \in \mathbb{G}$, the pairing $e(x, y)$ can be computed efficiently.*

There are several computational assumptions in the world of pairing-based cryptography. In this paper we will reduce the security of our protocol to the following:

Definition 2 (Decisional Linear (DLIN) Assumption [BBS04]). *Let $\mathbb{G}, \mathbb{G}_1$ be groups of prime order p with a bilinear map e as defined above. The decisional linear assumption, states that given three random generators f, h, g and f^r, h^s, g^t, it is hard to distinguish the case $t = r + s$ from a random t.*

3.2 Universally Composable Security Framework

If we want to prove that a protocol is secure, we firstly need to define what secure means. The universally composable security framework, defined by Canetti [Can01], is becoming a standard definition if one wants proper security guarantees. The strength of this framework relies in the universally composable theorem, that states that if a protocol is secure in the UC model, then this protocol will preserve the same security even if composed with an arbitrary number of copies of itself or with other protocols.

The price to pay for such a result is the impossibility of constructing any non-trivial protocol that is secure in the UC model[6]. In order to develop interesting protocols in the UC model we need some kind of setup assumptions. We put ourselves in the key registration (KR) authority scenario, first introduced in [BCNP04]. In this model, that has the flavor of a public-key infrastructure, we assume that there exists a trusted registration authority where parties can register public keys associated with their identities, while demonstrating that they have access to the corresponding secret keys. Alternatively, parties can let the authority choose public keys for them, in scenarios where the corresponding secret keys need not be revealed, even to the owners of the public keys. Then, parties can query the authority for a party identity and obtain the registered public key for that identity. Any ideal functionality can be UC realized by interactive protocols in the KR model, under standard computational hardness assumptions.

An advantage of the KR assumption, in respect to other setup assumptions like the common reference string model (CRS), is that it is trivial to ensure that all trust is not concentrated in one single entity. Namely, the receiver can register his key to several KRs, and the sender retrieve it from all of them. Now the sender only has to trust that one of the KRs does its job properly to be convinced that the receiver knows its secret key. A full comparison of the KR model against the CRS model is out of the scope of this paper, and we refer to [BCNP04] for more details.

There are several ways to implement a KR in the real world. In particular, as discussed in [BCNP04], it is possible to implement it with a stand-alone zero-knowledge proof of knowledge, if we have the guarantee that the proofs are run in a trusted environment — maybe the registrant shows up at the KR with the prover on a smartcard, and then the KR runs the smartcard in an isolated setting. If perfect isolation is not available, it is still possible to run an UC setup in the case of partial isolation [DNW08], where the parties are allowed to communicate with the environment, but just a limited amount of data.

We note that our protocol has *gracefully degradation*, as defined in [BCNP04]: if the proof of secret key given by the receiver is not UC (maybe because the assumption that the proof was running in an isolated

[6] Actually, it is possible to implement symmetric protocols like secure channels [CK02].

setting failed), but it is at least a stand-alone proof of knowledge, then our OT protocol too will be a stand-along secure implementation (of the multi-OT functionality) — the simulator will rewind the stand-alone proof of knowledge from the receiver to get the secret key, and then proceed as the UC simulator for all the OTs. Even if the proof fails to be just a stand-alone proof of knowledge, but it is at least a proof of membership, we will have some security, as the key being well-formed gives unconditional security for the sender, though without any composition guarantees. The implementation therefore in some sense delivers the best possible security level given the quality of the trusted setup.

4 Underlying Primitives

4.1 Mixed Commitment Scheme

We use a special kind of commitment called mixed commitment [DN02], which is a commitment scheme that can be instantiated with two kinds of key, giving two kinds of security. The first kind is perfectly binding and extractable, while the second is perfectly hiding and equivocal. A key which produces perfectly binding commitments will be called an extraction key (X-key) while a key that produces perfectly hiding commitments will be called an equivocal key (E-key). These two kinds of keys have to be computationally indistinguishable, in order for the commitment scheme to be called mixed.

We note that if we always instantiate the commitment scheme with X-keys, we end up with a commitment scheme that allows extraction, i.e. a public-key encryption scheme, where some keys (the E-keys) ensure that the "ciphertexts" contain no information about the plaintexts. We use this as an essential ingredient in our construction.

DLIN based Encryption Scheme. When we build our mixed commitment scheme, we start from the DLIN based cryptosystem from Boneh, Boyen and Shacham [BBS04], described now.

Let G be an algorithm that takes a security parameter as input and outputs $(p, \mathbb{G}, \mathbb{G}_1, e, g)$ such that p is prime, $\mathbb{G}, \mathbb{G}_1$ are descriptions of groups of order p, $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ is an admissible bilinear map, and g is a generator of $\mathbb{G}$. Those are the public parameters of the cryptosystem that works as follows:

Key Generation: Select x, y randomly in $\mathbb{Z}_p^*$, then compute $(f, h) = (g^x, g^y)$. The encryption key is $ek = (f, h)$ and the decryption key is $dk = (x, y)$.

Encryption: To encrypt a message $M \in \mathbb{G}$, select two random values $r, s \in \mathbb{Z}_p^*$. Then compute the encryption as $E_{ek}(M; r, s) = (\alpha, \beta, \gamma) = (f^r, h^s, g^{r+s}M)$.

Decryption: The message can be efficiently decrypted as $D_{dk}(\alpha, \beta, \gamma) = M = \alpha^{-1/x}\beta^{-1/y}\gamma$.

This encryption scheme is clearly IND-CPA secure under the DLIN assumption. Note that until now we did not use the pairing at all. The pairing will be used to prove statements about encryptions.

Mixed Commitment Scheme. We now describe the mixed commitment scheme. This is the commitment scheme under which the sender commits to x_0 and x_1 (under an E-key), and when instantiated with an X-key, it is the encryption scheme used by the receiver to encrypt b. This is an essential trick as it will make commitments and encryptions work together nicely.

The keys for the commitment scheme is going to be the ciphertexts of DLIN cryptosystem. I.e., a commitment key is of the form $ck = E_{ek}(M; r, s)$. A commitment to $m \in \mathbb{Z}_p$ is then of the form

$$\mathrm{Comm}_{ek,ck}(m; t, u) = ck^m E_{ek}(1; t, u) .$$

This is clearly homomorphic in the sense that

$$E_{ek}(M_0; t_0, u_0) E_{ek}(M_1; t_1, u_1) = E_{ek}(M_0 M_1; t_0 + t_1, u_0 + u_1) .$$

The basic idea of the mixed commitment scheme is then that if ck is an encryption of 1, then it follows that $ck^m E_{ek}(1; t, u)$ is a random encryption of 1. And, it is possible to efficiently open this commitment to any m given t and u and the randomness used to compute ck. On the other hand, if ck is the encryption of a generator (say g), then

$$ck^m E_{ek}(1; t, u)$$

is a random encryption of g^m, and it is therefore perfectly binding. In addition, it is possible to extract g^m from the commitment if the decryption key dk is known. Thanks to the properties of the cryptosystem, it is computationally infeasible to decide whether ck is an encryption of 1 or g, which is why it is a mixed commitment scheme.

This leads to the construction of our scheme as follows:

General key generation: There is a key pair (ek, dk), where $ek = (g, f, h) = (g, g^x, g^y)$ is an encryption key and $dk = (x, y)$ the decryption key. A full public key for the system is of the form (ek, ck), where ck is an encryption under ek.

Extraction key (X-key): For an X commitment key we have $ck = ck_X = E_{ek}(g; r, s) = (f^r, h^s, g^{r+s+1})$, where r and s are random. We will denote by $ck_X \leftarrow \mathrm{KG}_X$ the algorithm that produces an X-key, and we use $\mathcal{K}_X$ to denote the set of X-keys. The extraction trapdoor is $t_X = dk$.

Equivocal key (E-key): For an E commitment key we have $ck = ck_E = E_{ek}(1; r, s) = (f^r, h^s, g^{r+s})$, where r and s are random and $t_E = (r, s)$ is the equivocation trapdoor. We will denote by $ck_E \leftarrow \mathrm{KG}_E(r, s)$ the algorithm that produces an E-key, and we use $\mathcal{K}_E$ to denote the set of E-keys.

Committing: To commit to a message $m \in \mathbb{Z}_p$ under the general key ek and the commitment key ck, select $t, u \in_R \mathbb{Z}_p^*$, and compute $\mathrm{Comm}_{ek,ck}(m; t, u) = ck^m E_{ek}(1; t, u)$.

Opening: To open a commitment C, the committer releases (m, t, u). The receiver checks that $C = ck^m E_{ek}(1; t, u)$.

The general secret key dk allows to efficiently determine if the commitment key ck is an X-key or an E-key, otherwise, they will be indistinguishable from the properties of DLIN cryptosystem. Note that this commitment is homomorphic with respect to addition.

It is clear that the E-keys produce perfectly hiding commitments and that the X-keys produce perfectly binding commitments. Here is how equivocation and extraction work:

Equivocation: A random commitment to m under an E-key $ck \in \mathcal{K}_E$ is of the form

$$\begin{aligned}\mathrm{Comm}_{ek,ck}(m; t, u) &= ck^m E_{ek}(1; t, u)\\ &= (f^r, h^s, g^{r+s})^m (f^t, h^u, g^{t+u}) = (f^{rm+t}, h^{sm+u}, g^{rm+sm+t+u})\end{aligned}$$

for uniformly random t and u. Given any m' and letting $t' = r(m - m') + t$ and $u' = s(m - m') + u$, it follows that t' and u' are uniformly random and that

$$\begin{aligned}\mathrm{Comm}_{ek,ck}(m'; t', u') &= (f^{rm'+t'}, h^{sm'+u'}, g^{rm'+sm'+t'+u'})\\ &= (f^{rm+t}, h^{sm+u}, g^{rm+sm+t+u}) = \mathrm{Comm}_{ek,ck}(m; t, u)\ .\end{aligned}$$

I.e., given the randomness used to compute $\mathrm{Comm}_{ek,ck}(m; t, u)$ and the equivocation trapdoor $t_E = (r, s)$ of ck one can open $\mathrm{Comm}_{ek,ck}(m; t, u)$ to any value.

Extraction: For $ck \in \mathcal{K}_X$ and $c = \mathrm{Comm}_{ek,ck}(m; t, u) = ck^m E_{ek}(1; t, u)$ we have that $D_{dk}(c) = D_{dk}(ck)^m \cdot D_{dk}(E_{ek}(1; t, u)) = g^m$, from which m can be retrieved by exhaustive searching if it is from a small known set.

Note that the extraction has a limit on the size of m that can be extracted. In our first construction, our message set will be just $\{0, 1\}$ when we need extraction of m. In our second construction we consider $K = g^m$ to be a key and m just a value used to generate the key. In that case we can extract the key K for any m.

4.2 Efficient NIWI Proofs

Both during the choose and the transfer phase we need some non-interactive (NI) proofs. It turns out that these proofs do not have to be fully zero-knowledge. It is sufficient that they are witness indistinguishable (WI). Still, using standard WI proofs will result in increasing dramatically the number of rounds of the protocol. But also general NIWI proofs, even without increasing the number of rounds, will let the protocol be impractical. We use instead new NIWI constructions [GOS06, GS08] which allow to prove algebraic relations in bilinear groups. In particular, we use the following WI proofs:

Proof of Bit: In [GOS06] a composable NIWI to prove that the content of a commitment is either 0 or 1 is given. We will denote with $\pi_{0\vee 1}(c)$ the proof for the following relations: $R_{bit} = \{((ek, ck, c), (m, t, u)) | c = \mathrm{Comm}_{ek,ck}(m; t, u) \wedge m \in \{0, 1\}\}$. The proof consists of 6 group elements.

Proof of Multi-Exponent: In [GS08] a composable NIWI to prove the relation between the content of a number of commitments and the exponents of a multi-exponentiation is given. The proof consists of 2 group elements. We use it for 3 exponents and denote with $\pi_{MX}(c, g_1, g_2, g_3, C_1, C_2, C_3)$ the proof for the following relations:

$$\begin{aligned} R_{MX} = \{&((ek, ck, c, g_1, g_2, g_3, C_1, C_2, C_3), (x_1, x_2, x_3, t_1, t_2, t_3, u_1, u_2, u_3)) | \\ &c = g_1^{x_1} g_2^{x_2} g_3^{x_3}, \forall i = 1, 2, 3 : C_i = \mathrm{Comm}_{ek,ck}(x_i; t_i, u_i)\} \ . \end{aligned}$$

All the proofs are for the CRS model, where the proof assumes that a random common reference string crs has been honestly generated and it is known by the prover and the verifier. More precisely crs is sampled as $crs \leftarrow \mathrm{CRS}(r_{crs})$ for a poly-time algorithm CRS and uniformly random r_{crs}. In fact, there exist two different such generators CRS_S and CRS_Z. When crs is generated by CRS_S, then the proofs have *perfect* soundness and computational WI (under DLIN). When crs is generated by CRS_Z, then the proofs are *perfect* WI and computationally sound (under DLIN). The outputs of the two generators are in addition computationally indistinguishable. We can exploit this to avoid the CRS model at all. We let the receiver generate two common reference strings $crs_S = \mathrm{CRS}_S(r_{crs,S})$ and $crs_Z = \mathrm{CRS}_Z(r_{crs,Z})$ and use the KR authority to verify that crs_S and crs_Z were generated in this way. Then the proofs from the sender to the receiver are done under crs_Z, and the sender is guaranteed WI even if crs_Z was not generated at random, as the WI is perfect. Proofs from the receiver to the sender are done under crs_S, and the sender is guaranteed soundness even if crs_S was not generated at random, as the soundness is perfect.

5 Final Protocol

Since the mixed commitment scheme that we use does not have efficient extraction for arbitrary messages, we at first use it to construct *bit* OT. Later, we are discussing how to achieve *string* OT.

5.1 Parameter Agreement

We assume that all parties agree on the finite groups underlying the encryption scheme. In practice this would probably happen by the groups being described in some standard and hard-coded into the software for the OT module. In the UC model we model it using an ideal functionality which simply outputs a description of the groups to all parties. This ideal functionality can be thought of as the standardization body, and be activated with a message (get groups). It generates a DLIN group by running $(p, \mathbb{G}, \mathbb{G}_1, e, g) \leftarrow G(1^k)$, and outputs $param = (p, \mathbb{G}, \mathbb{G}_1, e, g)$ to all parties.

5.2 Key Registration Authority

Next we describe the registration phase. In the registration phase all parties which later want to act as receivers have to register a public key with a KR authority and prove knowledge of the corresponding secret key. Later all parties which want to act as senders can retrieve the public keys of the receivers from KR. Following [BCNP04] we model this simplistically by having one KR, by letting the registrants show knowledge of their secret keys by showing them directly to the KR, and letting the KR broadcast the corresponding public keys.

In more detail, the KR is parametrized by some poly-time relation R and accepts messages of the form $(\texttt{register}, pid, pk, sk)$ from some party P_i. It checks that $(pk, sk) \in R$ and if so sends (P_i, pk) to all parties. The relation R is chosen such that (pk, sk) being in R ensures that pk is a well-formed public key and that sk is the secret key.

In our protocol we use a public key of the form $pk = (param, ek, ck_X, ck_E, crs_Z, crs_S)$, where $param = (p, \mathbb{G}, \mathbb{G}_1, e, g)$ and $ek = (f, h)$, and we use a secret key of the form $sk = (dk, r_X, t_E, r_{crs,Z}, r_{crs,S})$, where $dk = (x, y)$. The relation R checks that $f = g^x$, $h = g^y$, $ck_X = \mathrm{KG}_X(r_X)$, $ck_E = \mathrm{KG}_E(t_E)$, $crs_Z = \mathrm{CRS_Z}(r_{crs,Z})$ and $crs_S = \mathrm{CRS_S}(r_{crs,S})$. I.e., it checks that ek is a well-formed public key for DLIN cryptosystem (and that the receiver knows the decryption key) and that ck_X is an X-key for our mixed commitment scheme and that ck_E is an E-key for our mixed commitment scheme (and that the receiver knows the equivocation trapdoor) and that crs_Z is a well-formed common reference string for the NIWI system giving perfect WI (and that the receiver knows how to simulate proofs), and that crs_S is a well-formed common reference string for the NIWI system giving perfect soundness.

The receiver P_i lets $param$ be the public parameters agreed upon by all parties, and it generates ek, ck_X, ck_E, crs_Z and crs_S at random, thereby learning the sk expected by KR. After key registration the receiver deletes r_X, t_E, $r_{crs,S}$ and $r_{crs,Z}$, as they are not needed in the protocol and they constitute a security risk if leaked. When the sender receives $(P_i, (param', ek, ck_X, ck_E, crs_Z, crs_S))$ it checks that $param'$ is equal to the parameters $param$ agreed upon earlier. If so, it remembers that the public key of receiver P_i is $(ek, ck_X, ck_E, crs_Z, crs_S)$.

5.3 1-out-of-2 Bit Oblivious Transfer

We now describe the communication phase. Here the parties can perform an unbounded number of OTs using the established PKI.

Choose: The receiver is given a bit b and an OT identifier $otid$. It computes a commitment $c_1 = (\alpha_1, \beta_1, \gamma_1) = \mathrm{Comm}_{ek,ck_X}(b)$. It sends $otid, c_1$ to the sender. It also sends $\pi_{0\vee 1}(c_1)$, computed under crs_S.

Transfer: The sender is given two secrets x_0, x_1 and $otid$. It waits for a message of the form $otid, c_1, \pi_{0\vee 1}$ from the receiver and checks the receiver's proofs and, if it accepts, it computes and sends to the receiver $otid, d = c_0^{x_0} c_1^{x_1} E_{ek}(1; r, s)$ with r and s chosen at random and with $c_0 = (\alpha_0, \beta_0, \gamma_0) = E_{ek}(1)c_1^{-1}$.[7] Note that when the sender is honest, then $d = (\alpha, \beta, \gamma) = (\alpha_0^{x_0}\alpha_1^{x_1} f^r, \beta_0^{x_0}\beta_1^{x_1} h^s, \gamma_0^{x_0}\gamma_1^{x_1} g^{r+s})$. In addition, the sender sends commitments $C_0 \leftarrow \mathrm{Comm}_{ek,ck_E}(x_0)$, $C_1 \leftarrow \mathrm{Comm}_{ek,ck_E}(x_1)$, $C_2 \leftarrow \mathrm{Comm}_{ek,ck_E}(r)$, $C_3 \leftarrow \mathrm{Comm}_{ek,ck_E}(s)$, and proofs $\pi_{MX}(\alpha, \alpha_0, \alpha_1, f, C_0, C_1, C_2)$, $\pi_{MX}(\beta, \beta_0, \beta_1, h, C_0, C_1, C_3)$ and $\pi_{MX}(\gamma, \gamma_0, \gamma_1, g, C_0, C_1, C_2C_3)$, to prove that C_0, C_1, C_2, C_3 commit to values x_0, x_1, r, s used to compute d. The NIWI proofs are performed under crs_Z.

[7] Here $E_{ek}(1)$ is some fixed encryption of 1 so that also R can compute c_0. This just saves the sending of c_0.

Retrieve: The receiver checks the proofs, and, if it accepts, it extracts d using dk and obtains $g^{x_b} = D_{dk}(d)$. If $g^{x_b} = 1$ it outputs 0, otherwise it outputs 1.

The protocol sends the 6 commitments $c_1, d, C_0, C_1, C_2, C_3$, each consisting of 3 group elements. Besides this a proof of size 6 is sent and 3 proofs of size 2 are sent, for a total of 30 group elements.

Theorem 1. *The protocol in Section 5.3 securely realizes F_{OT} in the UC-hybrid model.*

Proof: We address the case of the malicious sender and the malicious receiver separately:

Security against Malicious Receiver: The simulator runs the KR phase. If any of the keys registered by a corrupted receiver are not well formed, it ignores the registration, as would the real KR. Otherwise, it extracts from the receiver the secret key $(dk, r_X, t_E, r_{crs,Z}, r_{crs,S})$.

In the communication phase, when it receives a message $otid, c_1, \pi_{0\vee 1}$ from a corrupted receiver, it checks for the proof to be valid. If yes, it extracts $g^b = D_{dk}(c_1)$. If $g^b = 1$, it inputs 0 to the ideal functionality on behalf of the corrupted receiver. Otherwise it inputs 1. If the proofs fail, the simulator just ignores the message.

When the simulator is given x_b by the ideal functionality it behaves like an honest sender, after choosing a random x'_{1-b}.

This transfer message is perfectly indistinguishable from a protocol one. In fact d will be exactly the same compared to a real protocol run, being an encryption of the same message x_b. All the commitments C_1, C_2, C_3, C_4 have the same distribution, as ck_E is an E-key. The three proofs sent by the sender have the same distribution as the proofs are perfect WI when performed under crs_Z — in fact the receiver could have simulated the proofs itself.

Security against Malicious Sender: The simulator runs the KR phase. For all honest receivers it uses a random key $(ek, ck_X, ck_E, crs_Z, crs_S)$, where ck_X is a random E-key, ck_E is a random X-key, crs_Z is generated using CRS_S, and crs_S is generated using CRS_Z. These four are indistinguishable for the corrupted sender as E-keys and X-keys are indistinguishable and the output distributions of CRS_Z and CRS_S are indistinguishable.

In the communication phase, when it has to send a choose message for some $otid$ from the honest receiver to a corrupted sender, it does not know the real choice bit b. Instead it uses $b' = 0$.

When it receives $otid, d, C_1, C_2, C_3, C_4$ from the corrupted sender, along with the three proofs, it checks the proofs as in the protocol. If the proofs are valid, it extracts $g^{x_0} = D_{dk}(C_1)$ and $g^{x_1} = D_{dk}(C_2)$, and recovers x_0, x_1 by letting $x_i = 0$ if $g^{x_i} = 1$ and $x_i = 1$ otherwise, with $i = 0, 1$. Then it inputs them to the ideal functionality on behalf of the corrupted sender. Otherwise, it just ignores the message.

The Hybrid Argument. The above just sketches the simulator. Showing that the simulation with this simulator is indistinguishable from the real execution is as usual done using a hybrid argument.

That the simulation for a malicious receiver is perfect was already argued above. We therefore focus on the case of a malicious sender. We here sketch the sequence of hybrids used to go from the simulation to the real protocol.

In the first step the only change is to use $b' = b$ instead of $b' = 0$. Here b is the real choice bit, which is obtained by inspecting the ideal functionality — this the simulator cannot do, but we can do it as a mind spiel to define a hybrid distribution. Note that this step actually does not change the distribution as ck_X is an E-key in the simulation and the distribution of c_1 therefore does not depend on b' at all — it can be opened to both 0 and 1. Furthermore, the proof is perfect WI when crs_S is generated using CRS_Z, and the distribution of the proof therefore does not depend on which opening of c_1 is used.

In the second step we change ck_X to be a random X-key instead of a random E-key and generate crs_S using CRS_S. This change is indistinguishable to the corrupted parties as dk is kept secret by the honest receiver and the output distributions of CRS_S and CRS_Z are computationally indistinguishable. Note that after these two steps, the messages $c_1, \pi_{0\vee 1}$ have the same distribution as in the real protocol.

In the third step, the simulator computes the output x_b of the receiver, not by extracting x_0 and x_1 from C_1 and C_2 and inputting (x_0, x_1) to the ideal functionality (to make it output x_b), but by extracting a value x'_b from d and letting the ideal functionality output x'_b. When both ck_E and ck_X are X-keys, as they are at this point in the sequence of hybrids, it is straightforward to verify that if all three statements proved by the sender are true, it will always hold that $x'_b = x_b$. Since the NIWI proof system has perfect soundness when crs_Z is generated using CRS_S, as it is at this point in the sequence of hybrids, it follows that the third step actually makes no difference in the distribution.

In the last step ck_E is changed to be a random E-key instead of a random X-key and crs_Z is generated using CRS_Z. Again this is indistinguishable. Now the distribution is identical to the real protocol. $\square$

5.4 1-out-of-2 String Oblivious Transfer

In this section we present a protocol for string OT that is more efficient than achieving string OT by standard composition of bit OT.

The reason why we cannot just OT an n-bit value is that the receiver will not be able to decrypt the answer anymore. Recall that the value d sent from S to R is a commitment of the form $\mathrm{Comm}_{ek,ck_X}(x_b)$. This allows the receiver to efficiently compute g^{x_b} from d using the decryption key dk, as d is a commitment under the X-key ck_X. The problem is that if we let x_b be arbitrary, then computing x_b from g^{x_b} cannot be done efficiently.

Our idea is to consider our protocol a random OT, where the sender inputs two random values x_0, x_1, and the receiver gets a random group element $K_b = g^{x_b}$. Let us note that the sender cannot choose the values of K_0 and K_1, as the discrete logarithm problem is assumed to be hard here. However, he can compute himself the two elements K_0, K_1, as he knows g, x_0, x_1. At the end, S and R shares a random group element, that the sender can use as a key to encrypt his messages M_0, M_1, encoded as group elements, i.e. the sender computes and sends $X_0 = g^{x_0} M_0, X_1 = g^{x_1} M_1$ to the receiver. The receiver can retrieve $M_b = X_b K_b^{-1}$. Since K_{1-b} is a uniformly random group element completely unknown by the receiver, it gets no information on M_{1-b}. The only price paid is that we send two more group elements, bring the total communication up to 32 group elements.

Though intuitively clear, it remains to verify that the security is maintained. It turns out that the only non-trivial part is to check that the simulator can still extract the commitments from a malicious S — recall that ck_X is an E-key in the simulation and that the simulator therefore cannot extract d, but must extract the commitments given by the malicious S. We focus on this part, and leave the easier details to the reader.

Recall that e.g. the first component of d, being $\alpha_0^{x_0}\alpha_1^{x_1}\alpha^r$, is accompanied by three commitments $C_0 \leftarrow \mathrm{Comm}_{ek,ck_E}(x_0), C_1 \leftarrow \mathrm{Comm}_{ek,ck_E}(x_1), C_2 \leftarrow \mathrm{Comm}_{ek,ck_E}(r)$ and a proof that they commit to values used to compute the first component of d. Recall that in the simulation ck_E is an X-key. This means that the simulator can use dk to extract g^{x_0} from C_0 and g^{x_1} from C_1, which allows to compute $K_0 = g^{x_0}$ and $K_1 = g^{x_1}$, as desired. The proofs therefore, so to say, do not prove that S knows x_0 and x_1, but that S knows K_0 and K_1, which is sufficient to ensure that it knows M_0 and M_1.

Acknowledgments

We thank the anonymous reviewer from AFRICACRYPT 2008 and CRYPTO 2008 for the useful comments.

References

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2001.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195. IEEE Computer Society, 2004.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

[CNS07] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In Naor [Nao07], pages 573–590.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 581–596. Springer, 2002.

[DNW08] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Isolated proofs of knowledge and isolated zero knowledge. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 509–526. Springer, 2008.

[Fis06] Marc Fischlin. Universally composable oblivious transfer in the multi-party setting. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 332–349. Springer, 2006.

[Gar04] Juan A. Garay. Efficient and universally composable committed oblivious transfer and applications. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2004.

[GH07] Matthew Green and Susan Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In *ASIACRYPT*, pages 265–282, 2007.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008. `http://eprint.iacr.org/2007/155`.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie-hellman. In Wieb Bosma, editor, *ANTS*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In Naor [Nao07], pages 97–114.

[Kal05] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 78–95. Springer, 2005.

[Lin08] Yehuda Lindell. Efficient fully-simulatable oblivious transfer. In *CT-RSA*, 2008. `http://eprint.iacr.org/2008/035`.

[MOV93] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[Nao07] Moni Naor, editor. *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*. Springer, 2007.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.

[NP05] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *J. Cryptology*, 18(1):1–35, 2005.

[PVW07] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. Cryptology ePrint Archive, Report 2007/348, 2007. `http://eprint.iacr.org/`.

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. *Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981*, 1981.

[Wie83] Stephen Wiesner. Conjugate coding. *SIGACT News*, 15(1):78–88, 1983.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE, 1986.