Implementing Oblivious Transfer Using Collection of Dense Trapdoor Permutations

Iftach Haitner

Weizmann Institute of Science, Rehovot, Israel.

Abstract. Until recently, the existence of collection of trapdoor permutations (TDP) was believed (and claimed) to imply almost all of the major cryptographic primitives, including public-key encryption (PKE), oblivious transfer (OT), and non-interactive zero-knowledge (NIZK). It was recently realized, however, that the commonly accepted general definition of TDP needs to be strengthened slightly in order to make the security proofs of TDP-based OT go through. We present an implementation of oblivious transfer based on collection of dense trapdoor permutations. The latter is a collection of trapdoor permutations, with the property that the permutation domains are polynomially dense in the set of all strings of a particular length. Previous TDP-based implementations of oblivious transfer assumed an enhancement of the hardness assumption (of the collection).

1 Introduction

1.1 Oblivious Transfer (OT)

Oblivious transfer (OT), originally defined by Rabin [15], is a fundamental primitive in cryptography. OT has several equivalent formulations [15,7,3,5,2,4]. The version we studied, defined by Even, Goldreich and Lempel [7], is that of one-out-of-two OT. Informally, a (one-out-of-two) OT is a two-party protocol, in which one party (the `sender`) holds two secrets (σ_0 and σ_1) and the other party (the `receiver`) holds a secret bit i. If both parties follow the protocol, the `receiver` learns σ_i. In addition, even a cheating `receiver` (i.e., one that arbitrarily deviates from the protocol) cannot learn more then a single value in $\{\sigma_0, \sigma_1\}$ and even a cheating `sender` does not learn anything about i during the run of the protocol.

OT implies key agreement (KA) [15,1], signing contracts [7], and in general any secure multi-party evaluation [17,14,10].

1.2 Collection of Trapdoor Permutations (TDP)

A "collection of trapdoor permutations" (TDP) is among the strongest cryptographic primitives. TDP is a special case of collection of one-way permutations

M. Naor (Ed.): TCC 2004, LNCS 2951, pp. 394–409, 2004.

(OWP). Informally, a collection of permutations is one-way if a permutation chosen from this collection is easy to compute on any input, but hard to invert on the average. Any collection of OWP provides two additional efficient algorithms: The permutation sampler algorithm that samples a random permutation in the collection and the domain sampler algorithm that generates a random element in the domain of a given permutation. We stress that the permutation domains might be arbitrary, as long as there is an efficient domain sampler that generates a random element in them. Such a collection is called TDP, if in addition the permutation sampler algorithm produces a trapdoor information that allows its holder to invert the permutation. (see Subsection 3.4 for details).

1.3 Does TDP Implies OT?

Until recently, the existence of TDP was believed (and claimed) to imply OT. It was recently realized, however, that the commonly accepted general definition of TDP needs to be strengthened slightly in order to make the security proofs of TDP-based OT go through. This is due to the fact that in the standard TDP-based OT protocol, proposed by [7], the (honest-but-curious) `receiver` is expected to sample an element from the permutation domain such that the inverse of this element remains secret from its own point of view. The basic TDP security requirement guarantees secrecy against an external observer (who only observes the sampled element), however, the randomness used by the sampler could potentially be useful for efficient inversion. In fact, an arbitrary sampler could be used to construct a bad one, which first generates a domain element and then applies the permutation to produce the output.

To enable the stronger security feature required by the OT, Goldreich [12] defines a stronger primitive called "enhanced TDP". Specifically, an element produced by the domain sampler of an enhanced TDP should be hard to invert even when given the randomness used to produce it (see Subsection 3.5 for details). It should be noted that this distinction is quite hypothetical, as essentially all of the (very few) known TDP candidates can satisfy the stronger notion under the same assumptions.

1.4 Our Result

We show that OT can be based on any dense-TDP, where the latter is a TDP whose permutation domains are polynomially dense, i.e., contain polynomial fractions of the strings of length k (see Subsection 3.6 for details). We note that density assumption is made in other known constructions, such as the construction of non-interactive zero-knowledge proof of knowledge (NIZKPK) based on dense public-key crypto system [16].

1.5 Our Construction - Main Ideas

Our implementation follows the general ideas of the EGL protocol mentioned above. Recall that the EGL protocol is based on enhanced TDP (rather than a standard TDP) because the (honest-but-curious) `receiver` is expected to sample an element from the permutation domain such that the inverse of this element remains secret from its own point of view. In our construction, the `receiver` does not use the sampler, but rather chooses a random element in $\{0,1\}^n$ and checks whether or not the element is in the permutation domain. The main difficulty in our construction is the fact that it is not guaranteed that one can efficiently do the check above (i.e., check whether a given element is in the permutation domain). In order to overcome this difficulty, we extend a given dense-TDP into a dense-TDP with the extra property that there is an efficient algorithm that *given the permutation trapdoor* checks whether a given element in $\{0,1\}^n$ is inside the permutation domain. (We note that by doing this extension we are penalized, since the extended collection is not guaranteed to be dense-TDP but merely dense-weak-TDP , i.e., the collection's permutations are only guaranteed to be weak-one-way permutations). Then we use the latter TDP to implement a very weak form of OT, where we cannot assure that all the required properties of OT hold, but we can guarantee that they hold with a non-negligible probability. The above construction main idea is that the `sender` helps the `receiver` to check whether a given element is in the permutation domain. The final step of our construction is amplifying the above "weak OT" into a standard one. We note that even though amplifications of information theoretic weak forms of OT are quite common (e.g., [3,6]), amplifications of computational knowledge weak forms of OT, such as our amplification, are rare (in fact we encountered none) and are rather more complicated. Therefore, the amplification part of this paper may have a stand-alone value.

1.6 The Organization of the Rest of the Paper

In Section 2, we give a high level overview of our implementation. Section 3 is where we give the exact definitions of the tools and terms we use in this paper. In Section 4 we give the full implementation of a weak form OT based on dense-TDP and in Section 5 we show how to amplify such a "weak-OT" into a standard one.

2 Overview of Our OT

We present a polynomial time implementation of OT based on the existence of dense-TDP. Our implementation follows the general ideas of the following OT protocol, suggested by [7].

2.1 The EGL OT Protocol

Let (I, D, F) be a TDP, where I is the permutation sampler algorithm, D is the domain sampler algorithm and F and F^{-1} are the evaluation and inverting algorithms respectively (see Subsection 3.4 for details). Recall that the protocol's inputs are: the `sender`'s secrets, σ_0 and σ_1, the `receiver`'s index, i and the security-parameter, n, given in unary.

1. The `sender` uniformly selects a permutation description, α, along with its trapdoor, t, by letting $(\alpha, t) \leftarrow I(1^n)$.
   The `sender` sends α to the `receiver`.
2. The `receiver` uniformly selects two elements, r_0 and r_1, in D_α, as follows: r_{1-i} is selected directly in D_α, using the sampler, D. In order to select r_i, the `receiver` selects a third element, s, in D_α (using the sampler) and then sets r_i to $f_\alpha(s)$.
   Hence, the `receiver` knows the pre-image of r_i (i.e., s), but does not know the pre-image of r_{1-i}. Note that since f_α is a permutation, both r_0 and r_1 have the same distribution and thus, knowing them gives no information about i.
   The `receiver` sends (r_0, r_1) to the `sender`.
3. For both $j = 0, 1$, the `sender` computes $c_j = \sigma_j \oplus b(f_\alpha^{-1}(r_j))$, where b is a hardcore predicate for f_α.
   The `sender` sends (c_0, c_1) to the `receiver`.
4. The `receiver` locally outputs $c_i \oplus b(s)$ (and as $c_i \oplus b(s) = c_i \oplus b(f_\alpha^{-1}(r_i)) = \sigma_i$, it outputs σ_i).
   Note that as the `receiver` does not know the value of $f_\alpha^{-1}(r_{1-i})$, it received no knowledge about σ_{1-i}.

The security of the above protocol relies on the fact that the `receiver` does not know the pre-image of r_{1-i}, even though the `receiver` knows the random coins used by the sampler to select r_{1-i}. Therefore, the above protocol requires that the TDP be an enhanced one.

2.2 Towards the Protocol

We call a given TDP "checkable-domains-TDP", if there is an efficient algorithm that checks whether a given element in $\{0,1\}^n$ is inside the permutation domain (clearly, a given TDP might not have this property). We start by showing how to implement an OT based on checkable-domains-dense-TDP, and then step-by-step, show how to implement an OT based on a standard dense-TDP.

An OT Based on Checkable-Domains-Dense-TDP. The protocol follows the same lines as the EGL protocol (described in Subsection 2.1), except for Step 2 that has the following form:
2. The `receiver` selects s, r_i and r_{1-i} as follows:

a. s and r_{1-i} are chosen uniformly in $\{0,1\}^n$.
b. The `receiver` checks whether both s and r_{1-i} are in D_α. If the answer is negative, the `receiver` restarts the protocol (the two parties go back to the first step of the protocol).
c. r_i is set to $f_\alpha(s)$.

It is easy to see that the above construction is indeed an implementation of OT [1]. We stress that since the `receiver` did not use the collection sampler to choose r_{1-i}, the above is still true even if the collection is not enhanced.

Our next step is to implement a dense-TDP based OT with a weaker property than the checkable-domains one. We call a given TDP a "t-checkable-domains-TDP", if there is an efficient algorithm that *given the permutation trapdoor* checks whether a given element in $\{0,1\}^n$ is inside the permutation domain. We do not construct an OT based on t-checkable-domains-dense-TDP directly, but rather construct some weak form of OT. We shall later show how this weak form of OT can be amplified into a standard OT.

2.3 A Weak OT Based on T-Checkable-Domains-Dense-TDP

The first idea is to try and use a similar protocol to Protocol 2.2, where in order to decide whether or not s and r_{1-i} are in D_α, the `receiver` sends both elements to the `sender` in random order, and the `sender` (using the trapdoor) does the check and returns the answer to the `receiver`. If the `sender`'s answer is positive, then the `receiver` sends $f_\alpha(s)$ and r_{1-i} to the `sender` and the protocol proceeds as in Protocol 2.2, otherwise the `receiver` restarts the protocol. It is easy to see, however, that this protocol leaks the value of i to the `sender` (as the `sender` gets both s and $f(s)$)).

A better idea is for the `receiver` to send the `sender` $f_\alpha(s)$ [2] and r_{1-i} (instead of s and r_{1-i}) in random order and the `sender` answers whether both elements are in D_α. Only if the `sender`'s answer is positive, the `receiver` reveals the right order of $f_\alpha(s)$ and r_{1-i}, and the protocol proceeds as in Protocol 2.2. At first glance it seems as thought we have a solution; unfortunately this is not the case, as it turns out that not only information about i might leak, but also the `receiver` might miscalculate the value of σ_i. The problem is that even if $f_\alpha(s)$ is in D_α, we are not guaranteed that s is. The reason is that f, when extended to $\{0,1\}^n$, is not necessarily a permutation and therefore s might be outside D_α even if $f_\alpha(s)$ is in D_α. Therefore the `receiver` might miscalculate the value of σ_i. Moreover, as f is not a permutation on $\{0,1\}^n$, $f_\alpha(s)$ and r_{1-i} might have a different distribution and hence, by revealing them to the `sender`, some information about i might leak.

Fortunately, there is a way to overcome the problems above, or more accurately to ensure that the constructed protocol is some weak form of OT. (By a weak form of OT, we mean that even though we cannot assure that all the required properties of OT hold, we can guarantee that they hold with non-negligible probability). The solution is that in addition to checking whether both elements (i.e., $f_\alpha(s)$ and r_{1-i}) are in D_α, the `sender` sends to the `receiver`

---

[1] There is a subtle point regarding the running time of the above protocol, which is not even guaranteed to stop. Due to the density property of the collection, however, this issue can be easily solved.

[2] By $f_\alpha(x)$, where x is not guaranteed to be in D_α, we mean the result of invoking the collection evaluating algorithm, F, with inputs α and x.

some random information [3] about the pre-images (with respect to f_α) of the two elements. The receiver checks whether the information it received about the pre-image of $f_\alpha(s)$ is consistent with s. If the answer is negative (or if one of the elements is not in D_α) it restarts the protocol. By keeping the amount of information the sender sends about pre-images small [4], we guarantee that only small amount of information about the pre-image of r_{1-i} (and therefore about σ_{1-i}) has leaked to the receiver. On the other hand, even though the amount of information is limited, we can guarantee with sufficiently high probability (which depends on the amount of information sent and the density of the collection) that the chosen s is indeed the pre-image of r_i. Hence, the protocol is a weak form of OT where all the required properties hold with sufficiently high probability [5].

We are now ready to construct a "very" weak form of OT (even weaker than the above) based on dense-TDP without any other assumptions.

2.4 A "Very" Weak OT Based on Any Dense-TDP

The main idea is that any dense-TDP can be extended into a t-checkable-domains-dense-TDP. (We note that by doing this extension we are penalized, since the extended collection is not guaranteed to be dense-TDP but merely dense-weak-TDP , i.e., the collection's permutations are only guaranteed to be weak-one-way permutations). The construction of the extended collection is as follows. For each permutation f_α with domain D_α of the original collection, the extended collection has the permutation f'_α with domain D'_α. Where $D_\alpha \subseteq D'_\alpha \stackrel{\text{def}}{=} \left\{x \in \{0,1\}^n \,|f_\alpha(\alpha, (f_\alpha^{-1}(x)) = x\right\}$ and f'_α is defined to be the natural extension of f_α to D'_α, that is $f'_\alpha(x) \stackrel{\text{def}}{=} F(\alpha, x)$.

By the density property of the collection we have that for any given permutation α, $\frac{|D_\alpha|}{|D'_\alpha|}$ is not negligible, therefore the extended collection's permutations are weak-one-way permutations. Hence, the extended collection is a dense-weak-TDP. Moreover, given an element x in $\{0,1\}^n$ and a permutation trapdoor, one can easily check whether x is in the permutation domain by checking whether $f_\alpha(\alpha, (f_\alpha^{-1}(x)) = x$.

By using Protocol 2.3 with the above dense-weak-TDP as the underlying collection, we construct some weak form of OT. This form of OT is even weaker than the one achieved by Protocol 2.3 as the collection's permutations are only weak one-way and hence, some information about σ_{1-i} might leak to the receiver through the run of the protocol. Nevertheless, this weaker form can still be amplified into a standard OT.

---

[3] In our implementation the random information is the output of applying a randomly chosen pairwise independent hash function on the pre-images.

[4] By small amount of information we mean *polylog*(n) bits of information, where n is the security-parameter of the protocol.

[5] Actually the secrecy of the other secret (i.e., σ_{1-i}) is guaranteed with probability 1.

2.5 The Amplification Step

The amplification of the above "very" weak OT into a standard OT, is done in three consecutive steps. In each step we amplify a different property of the protocol. Hence, after the third step we have a standard OT.

3 Definitions

3.1 The Semi-honest Model

Loosely speaking, a semi-honest party is one that follows the protocol properly with the exception that it keeps a record of all its intermediate computations. In the semi-honest model all parties are assumed to be semi-honest. As far as the implementation of cryptographic protocol is concerned, one can limit oneself to the semi-honest model. The reason is that in [10] it is shown that semi-honest model protocols can be extended to the general (malicious) model, in which nothing is assumed regarding the parties. (For details see [12]).

3.2 Oblivious Transfer (OT)

A (one-out-of-two) OT is a two-party protocol, it has three inputs: the `sender`'s secrets, σ_0 and σ_1, and the `receiver`'s index, i in $\{0, 1\}$. In addition, the protocol receives, as an input, its security-parameter, n, given in unary [6]. The OT has the following properties:

1. Correctness - The `receiver` almost always learns σ_i. That is, the `receiver` learns σ_i with probability at least $1 - neg(n)$ ($neg(n)$ stands for negligible function of n), where the probability is over both parties' internal coin tosses.
2. `Sender`'s privacy - The `receiver` gains no computational knowledge about σ_{1-i}. More formally, let $VIEW_R(\sigma_i, \sigma_{1-i}, i)$ be the random variable defined from the `receiver`'s view of the protocol where σ_i and σ_{1-i} are the `sender`'s input and i is the `receiver`'s input [7]. Then for any polynomial time algorithm M, for any choices of σ_i, i and large enough n,
$$|Pr[M(VIEW_R(\sigma_i, 1, i)) = 1] - Pr[M(VIEW_R(\sigma_i, 0, i)) = 1]| < neg(n)$$
where the probability is over both parties' internal coin tosses.
3. `Receiver`'s privacy- The `sender` gains no computational knowledge about i.

In this paper we focus in on implementing OT in the semi-honest model (see Subsection 3.1 for details), which by [10] yields an implementation in the general (malicious) model. Furthermore, we limit ourselves to OT whose secrets are one bit long. Implementing this limited version suffices, as by successive use of one bit protocol we construct the non-limited version.

[6] We usually omit the security-parameter from the protocol's input parameters list.

[7] The above notation is somewhat misused, as the order of the parameters depends on their values. Nevertheless, the underlying notation is clear, and it is done for the sake of simplicity.

3.3 $(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$

$(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$ is a two-party protocol that serves as an intermediate step in our implementation of OT. $(\epsilon_1, \epsilon_2, \epsilon_3)$–$WOT$ is a relaxed version of OT. Whereas in OT it is required that no knowledge except for the required secret may leak from one party to the other, in $(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$ some amount of knowledge might leak (ϵ_2 is the amount of knowledge that might leak from the `sender` to the `receiver` and ϵ_3 is the amount of knowledge that might leak from the `receiver` to the `sender`). Furthermore, even the value of the required secret is not guaranteed to pass correctly (it is only guaranteed to pass with probability $1 - \epsilon_1$). Thus the ϵ's measure the weaknesses of the protocol and the smaller they are the better the protocol is. Let us turn to the formal definition.

$(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$ is a two-party protocol, it has three inputs: the `sender`'s secrets, σ_0 and σ_1 in $\{0, 1\}$, and the `receiver`'s index, i in $\{0, 1\}$. In addition, the protocol receives, as an input, its security-parameter, n, given in unary. $(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$ has the following properties:

1. Correctness - The `receiver` learns σ_i with probability at least $1 - \epsilon_1$, where the probability is over both parties' internal coin tosses.
2. `Sender`'s privacy - The `receiver` does not gain more **computational knowledge** about σ_{1-i} than ϵ_2. More formally, let $VIEW_R(\sigma_i, \sigma_{1-i}, i)$ be the random variable defined from the `receiver`'s view of the protocol where σ_i and σ_{1-i} are the `sender`'s input and i is the `receiver`'s input. Then for any polynomial time algorithm M, for any choices of σ_i, i and large enough n,
$$|Pr[M(VIEW_R(\sigma_i, 1, i)) = 1] - Pr[M(VIEW_R(\sigma_i, 0, i)) = 1]| < \epsilon_2$$
where the probability is over both parties' internal coin tosses.
3. `Receiver`'s privacy - The `sender` does not gain more **information** about i than ϵ_3. More formally, let $VIEW_S(\sigma_i, \sigma_{1-i}, i)$ be the random variable defined from the `sender`'s view of the protocol where σ_i and σ_{1-i} are the `sender`'s input and i is the `receiver`'s input. Then for any choices of σ_i and σ_{1-i} and large enough n,
$$stat(VIEW_S(\sigma_i, \sigma_{1-i}, 1), VIEW_S(\sigma_i, \sigma_{1-i}, 0)) < \epsilon_3$$

Note that in the above definition, the third parameter (`Receiver`'s privacy) measures information rather than computational knowledge. This strengthening simplifies our construction, as information theoretic reductions are much simpler than computational knowledge reductions.

3.4 Collection of Trapdoor Permutations (TDP)

Collection of trapdoor permutations (uniform complexity version) [11]: Let $\overline{I} \subseteq \{0, 1\}^*$ and $\overline{I}_n \stackrel{\text{def}}{=} \overline{I} \cap \{0, 1\}^n$. A collection of permutations with indices in $\overline{I}$ is a set $\{f_i : D_i \rightarrow D_i\}_{i \in \overline{I}}$ such that each f_i is one-to-one on the corresponding D_i. Such a collection is called a trapdoor permutation is there exist four probabilistic polynomial-time algorithms I, D, F, F^{-1} such that the following five conditions hold:

1. $Pr[I(1^n) \in \overline{I}_n \times \{0,1\}^*] > 1 - 2^{-n}$.
   That is, I is used to generate a random permutation along with its trapdoor.
2. Selection in domain, for every $n \in \mathbb{N}$ and $i \in \overline{I}_n$
   a) $Pr[D(i) \in D_i] > 1 - 2^{-n}$.
   b) Conditioned on $D(i) \in D_i$, the output is uniformly distributed in D_i. Thus $D_i \subseteq \cup_{m \leq poly(|i|)} \{0,1\}^m$. Actually, $D_i \subseteq \{0,1\}^{poly(|i|)}$.

   That is, given a permutation, D is used to generate a random element in the permutation domain.
3. Efficient evaluation, for every $n \in \mathbb{N}, i \in \overline{I}_n$ and $x \in D_i, Pr[F(i,x) = f_i(x)] > 1 - 2^{-n}$.
   That is, given a permutation, F is used to evaluate the permutation on any element in its domain.
4. Hard to invert, let I_n be the random variable describing the distribution of the first element in the output of $I(1^n)$ and $X_n \stackrel{\text{def}}{=} D(I_n)$, thus, for any polynomial time algorithm M, every polynomial p, and large enough n, $Pr[M(I_n, f_{I_n}(X_n)) = X_n] < \frac{1}{p(n)}$.
5. Inverting with trapdoor, for every $n \in \mathbb{N}$ any pair (i,t) in the range of $I(1^n)$ such that $i \in \overline{I}_n$, and every $x \in D_i$, $Pr[F^{-1}(t, f_i(x)) = x] > 1 - 2^{-n}$.
   That is, given a permutation along with its trapdoor, F^{-1} is used to find the pre-image of any element in its domain.

3.5 Enhanced Collection of Trapdoor Permutations

The implementation of OT presented by [7], is based on the existence of enhanced collection of trapdoor permutations. The enhancement refers to the hard-to-invert condition; i.e., it is hard to find the pre-image of a random element without knowing the permutation trapdoor. The enhanced condition requires that the hardness still hold *even when the adversary receives, as an additional input, the random coins used to sample the element.* (For more details see [12]).

It is presently unknown whether or not the existence of a TDP implies the existence of an enhanced TDP.

3.6 Collection of Dense Trapdoor Permutations (Dense-TDP)

A collection of dense trapdoor permutations (dense-TDP) is a TDP with one additional requirement. Whereas in an arbitrary TDP, the permutations may have arbitrary domains, here we require that these domains be polynomial fractions of the set of all strings of a particular length. Formally, let D_α be the domain of the permutation named α, then the additional requirement is: There exists a positive polynomial g such that for all $n \in \mathbb{N}$ and all $\alpha \in \overline{I}_n$, $D_\alpha \subseteq \{0,1\}^n$ and $|D_\alpha| > \frac{2^n}{g(n)}$. We define the density parameter of the collection, ρ, as $\frac{1}{g}$.

An alternative definition might allow D_α to be a subset of $\{0,1\}^{k(n)}$, for some fixed positive polynomial k (rather than a subset of $\{0,1\}^n$). It is easy to see, however, that the two definitions are equivalent.

4 Using Dense-TDP to Construct $\left(\frac{1}{q(n)}, 1 - \frac{\rho(n)^2}{4}, \frac{1}{q(n)}\right) - WOT$

(where ρ is the density parameter of the collection and q is any positive polynomial)

In this section we implement a very weak form of $(\epsilon_1, \epsilon_2, \epsilon_3) - WOT$, as all three parameters are not negligible. Notice that while the second parameter is fixed (equals $1 - \frac{\rho(n)^2}{4}$) and might be rather big, the first and third parameters can be as small as we like (as long as they are polynomial fractions). This freedom in choosing the first and third parameters, is used by the next section in order to construct a stronger protocol.

4.1 Preliminaries

Let (I, D, F) be a dense-TDP with density parameter ρ. For simplicity's sake, we assume that the collection's algorithms are deterministic and errorless, i.e., always return the right answers. Note that in the definition of dense-TDP, the algorithms are probabilistic and might return wrong answers with negligible probability. Extending the following implementation to the general case, however, is rather straightforward. (For details see [13]).

We would like to evaluate $F(\alpha, \cdot)$ and $F^{-1}(\alpha, \cdot)$ on any element in $\{0,1\}^n$ (and not only on elements in D_α). The problem is that nothing is guaranteed about the computation of $F(\alpha, x)$ and $F^{-1}(\alpha, x)$ when x is not in D_α. We can assume, however, that this computation halts in polynomial time and returns some value in $\{0,1\}^n$. Therefore we extend the notations $f_\alpha(x)$ and $f_\alpha^{-1}(x)$ to denote, for all $x \in \{0,1\}^n$, the value of $F(\alpha, x)$ and $F^{-1}(\alpha, x)$ respectively.

4.2 The Protocol's Outline

This protocol is an extension of the EGL protocol (described in Subsection 2.1). The first part of the protocol (Steps 1-3) is similar to the first part (Steps 1-2) of the EGL protocol. In this part, the `receiver` selects r_{1-i} and s uniformly in $\{0,1\}^n$. Note that either r_{1-i} or s might not be in D_α. This fact reduces the protocol's quality and hence, we are not constructing (in this step) an OT. There is a non-negligible probability, however, that both elements are in D_α, and therefore the protocol can guarantee some weaker requirements.
The middle part of the protocol (Steps 4-5) is where the new key idea lies. The `sender` helps the `receiver` to decide whether or not r_0 and r_1 (that the `receiver` has chosen in the first part of the protocol) "look" as though they have been chosen from the same distribution. In addition, the `sender` helps the `receiver` to decide whether or not s is the pre-image (with respect to f_α) of r_i. The above help is given to the `receiver` without leaking "too much" information about the value of σ_{1-i}. This help is needed, as there is no efficient way to decide whether or not a given element is in D_α. If the `receiver` concludes that r_0 and r_1 "look" as though they have been chosen from different distributions, or that

s is not the pre-image of r_i, then it restarts the protocol. Hence, the protocol might iterate through its first two parts (Steps 1-5) for quite a while, before it finally reaches its last part (Steps 6-8). It is guaranteed, however, that with very high probability, the protocol halts after a polynomial number of iterations.
The last part of our protocol is similar to the last part (Steps 3-4) of the EGL protocol. The `receiver` uses the information it received from the `sender` to calculate σ_i. Note that when s is in D_α (which happens with probability at least ρ), the `receiver` receives the right value of σ_i.

4.3 The Protocol

The protocol uses a collection of pairwise independent hash functions denoted H_n, where the hash function domain is $\{0,1\}^n$ and their range is $\left\{1, 2, \ldots, \frac{q(n)}{\rho^2(n)}\right\}$ [8]. Recall that the protocol's inputs are: the `sender`'s secrets, σ_0 and σ_1, and the `receiver`'s index, i.

1. The `sender` uniformly selects a permutation and its trapdoor, α and t, by letting $(\alpha, t) \leftarrow I(1^n)$, and uniformly selects a hash function $h \in H_n$.
   The `sender` sends (h, α) to the `receiver`.
2. The `receiver` selects s, r_i and r_{1-i} as follows:
   - s is chosen uniformly in $\{0,1\}^n$ and r_i is set to $f_\alpha(s)$.
   - r_{1-i} is chosen uniformly in $\{0,1\}^n$.

   The idea is that when s is in D_α, the `receiver` knows the value of $f_\alpha^{-1}(r_i)$ (i.e., s), and when r_{1-i} is in D_α, it does not know the value of $f_\alpha^{-1}(r_{1-i})$. Moreover, when both s and r_{1-i} are in D_α, they have the same distribution (as f_α is a permutation on D_α) and thus, knowing them gives no knowledge about i. Note that, if r_{1-i} or s are not in D_α, then the protocol is not guaranteed to work correctly, but with sufficiently high probability, the protocol detects such a situation by itself (Steps 4 - 5).
3. The `receiver` sends $(r_0,\ r_1)$ to the `sender` in random order, i.e., the `receiver` selects k uniformly in $\{0,1\}$, sets w_0 to r_k and w_1 to r_{1-k}, and sends (w_0, w_1) to the `sender`.
   By sending r_0 and r_1 in random order, the `receiver` hides the identity of i. The random order is needed, since r_0 and r_1 might have completely different distributions and thus, sending them in a fixed order might leak information about i. This random ordering step was not taken in the EGL protocol, as in the EGL protocol both r_0 and r_1 were guaranteed to have the same distribution (recall that they were uniformly chosen in D_α). In the current protocol, however, it is not always the case. The reason is that in order to choose r_i we evaluate $f_\alpha(s)$, even though s is not guaranteed to be in D_α. Hence, we can assure nothing about r_i distribution. For example, it might be that for all x not in D_α, $f_\alpha(x)$ equals 0^n.

---

[8] That is, for any n, for any $x, y \in \{0,1\}^n$ and for any $\alpha, \beta \in \left\{1, 2, \ldots, \frac{q(n)}{\rho^2(n)}\right\}$ $Pr_{h \in_R H_n}[(h(x) = \alpha) \wedge (h(y) = \beta)] = \left(\frac{\rho(n)^2}{q(n)}\right)^2$.

4. For both $j = 0, 1$, the `sender` checks whether $f_\alpha(f_\alpha^{-1}(w_j))$ is equal to w_j. If the answer is positive it sets v_j to $h(f_\alpha^{-1}(w_j))$, otherwise it **aborts** the current iteration (i.e., the protocol is restarted).
The `sender` sends (v_0, v_1) to the `receiver`.
5. The `receiver` **aborts** the current iteration, if $v_{i \oplus k}$ is not equal to $h(s)$.
Motivating comment for Steps 4 and 5: The goal of the last two steps is to ensure, with sufficiently high probability, that the following two requirements hold. The first requirement is that s is the pre-image of r_i and the second requirement is that r_i and r_{1-i} "look" as though they have been chosen from the same distribution. The crucial observation is that when both r_i and r_{1-i} happen to be in D_α (which happens with probability at least $\frac{1}{\rho(n)^2}$), then the above two requirements are guaranteed to hold and the current iteration is not aborted. On the other hand, when one of the above two requirements does not hold, then the current iteration is aborted with probability at least $1 - \frac{1}{q(n)}$. (When s is not the pre-image of r_i then the `receiver` detects it in Step 5 with probability $1 - \frac{1}{q(n)}$, and when s is the pre-image of r_i and the current iteration is not aborted, then both r_i and r_{1-i} are uniformly distributed in the set $D'_\alpha \stackrel{\text{def}}{=} \{x \in \{0,1\}^n \,| f_\alpha(\alpha, (f_\alpha^{-1}(x)) = x\})$.
We note that even though some information about the pre-image of r_{1-i} (i.e., $v_{(1-i)\oplus k}$) is delivered to the `receiver`, this information is given in a small amount and thus does not enable the `receiver` to compute the pre-image of r_{1-i} by itself.
6. The `receiver` sends k to the `sender`.
That is, the `receiver` tells the `sender` which of the values, w_0 and w_1, is r_0 and which is r_1. The point is that when we reach this step, r_0 and r_1 have, with substantial probability, the same distribution. Hence, only a small amount of information about i might leak to the `sender`.
7. For both $j = 0, 1$, the `sender` uniformly chooses $y_j \in \{0,1\}^n$ and sets c_j to $b(f_\alpha^{-1}(r_j), y_j) \oplus \sigma_j$, where $b(x, y) \stackrel{\text{def}}{=} < x, y > \bmod 2$ (i.e., the inner product of x and y modulus 2).
The `sender` sends (c_0, c_1, y_0, y_1) to the `receiver`.
Note that in this protocol, the `sender` XORs σ_0 and σ_1 with the hardcore bits of (r_0, y_0) and (r_1, y_1). The latter hardcore bits are with respect to a specific hardcore predicate (i.e., b) of the trapdoor permutation g_α, defined as $g_\alpha(x, y) \stackrel{\text{def}}{=} (f_\alpha(x), y)$. Recall that in the EGL protocol, the `sender` XORs σ_0 and σ_1 with the hardcore bits of r_0 and r_1, with respect to any given hardcore predicate of $f\alpha$. The reason for this modification is that in our proof of security, we rely on the structure of the above specific hardcore predicate.
8. The `receiver` locally outputs $b(s, y_i) \oplus c_i$.
Note that when $f_\alpha^{-1}(r_i)$ is equal to s, the `receiver` outputs σ_i. In addition, when r_{1-i} is in D_α, no knowledge about σ_{1-i} leaks to the `receiver`.

Analysis Sketch. In this section we sketch the correctness proof of the above protocol. (A detailed proof can be found at [13]).

We first mention that the protocol is a polynomial one. The reason is that by the density property of the collection, the protocol halts, with very high probability, after $\frac{n}{\rho(n)^2}$ iterations. Thus, without loss of generality, we can assume that the protocol always halts after $\frac{n}{\rho(n)^2}$ iterations and is therefore polynomial.

We say that the protocol had a good-ending, if in its last iteration s was equal to the pre-image of r_i (with respect to f_α). It is not hard to see that the probability for a good-ending of the protocol, is at least $1-\frac{1}{q(n)}$. The proof of the protocol's first and third properties (i.e., Correctness and Receiver's privacy), is a direct implication of the above result.

The proof of the protocol's second property (Sender's privacy), is proved by contradiction. We assume that the second property does not hold, and construct a polynomial time algorithm that, with non-negligible success, inverts the collection of dense trapdoor permutations. The proof has two major steps. First we construct a polynomial time algorithm, B, that predicts $b(f_\alpha^{-1}(x), y)$ with non-negligible advantage. (Recall that $b(z, w)$ is the inner product of z and w mod 2, and it is a hardcore predicate of the trapdoor permutation g_α, defined as $g_\alpha(z, w) \stackrel{\text{def}}{=} (f_\alpha(z), w))$. In the second step, we construct a polynomial time algorithm that computes $f_\alpha^{-1}(x)$, by embedding B in the reduction given by [9] in proving *hard-core predicate for any one-way function.*

5 Amplifying $\left(\frac{1}{q(n)}, 1-\frac{\rho(n)^2}{4}, \frac{1}{q(n)}\right)-WOT$ to OT

(where ρ is the density parameter of the collection and we have the freedom to choose q as any positive polynomial)

In this section we sketch how to amplify a general $\left(\frac{1}{q(n)}, 1-\frac{1}{t(n)}, \frac{1}{q(n)}\right)$–$WOT$ to OT, where t is any positive polynomial (not necessarily $\frac{4}{\rho(n)^2}$) and we have the freedom to choose q as any positive polynomial. A detailed construction can be found at [13].

The amplification is done in three independent steps. Each step amplifies some weak form of OT into a stronger form. In Subsection 5.4 we show how to combine these steps to create the desired amplification.

5.1 Using $\left(\frac{1}{nq'(n)t(n)}, 1-\frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)-WOT$ to Construct $\left(\frac{1}{q'(n)}, neg(n), \frac{1}{q'(n)}\right)-WOT$

(where q' and t are any positive polynomials)

In this step, we show how to reduce (the potentially big) second parameter of $\left(\frac{1}{nq'(n)t(n)}, 1-\frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)$–$WOT$ into a negligible function. In the protocol, the sender splits its original pair of secrets into many pairs of secrets, by splitting each of the original secrets into many secrets using a secret sharing scheme. Then, the sender transfers the i'th secret of each new pair to the receiver using $\left(\frac{1}{nq'(n)t(n)}, 1-\frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)$–$WOT$. The point is that in order to know the value of σ_j, one should know the j'th secret of each of the new pairs. Thus the amount of knowledge the receiver gains about σ_{1-i} in the following protocol is negligible.

The Protocol. Recall that the protocol's inputs are: the sender's secrets, σ_0 and σ_1, and the receiver's index, i.

1. For both $j = 0, 1$, the sender sets the following values:
   - $\omega_{j,1}, \dots, \omega_{j,nt(n)-1}$ are uniformly chosen at $\{0,1\}$.
   - $\omega_{j,nt(n)}$ is set to $(\bigoplus_{k=1}^{nt(n)-1} \omega_{j,k}) \oplus \sigma_j$.
2. For all $1 \le k \le nt(n)$, the sender transfers $\omega_{i,k}$ to the receiver, using $\left(\frac{1}{nq'(n)t(n)}, 1 - \frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)-WOT$.
3. The receiver locally outputs $\bigoplus_{k=1}^{nt(n)} \omega_{i,k}$.

Analysis Sketch. By invoking Yao's XOR lemma, we show that the amount of knowledge the receiver gains about σ_{1-i} through this protocol is negligible and hence, the Sender's privacy is guaranteed. Proving the other properties (Correctness and Receiver's privacy) is rather straightforward. The Correctness property is proved by analyzing the probability that all the invocations of the subprotocol are successful (in a sense that the receiver always computes the right value of $\omega_{i,\cdot}$), and the proof of the Receiver's privacy property is proved by a simple information theoretic argument.

5.2 Using $\left(\frac{1}{nq''(n)}, neg(n), \frac{1}{nq''(n)}\right) - WOT$ to construct $\left(neg(n), neg(n), \frac{1}{q''(n)}\right) - WOT$

(where q'' is any positive polynomial)

In this step, we show how to reduce the first parameter into a negligible function. In the protocol the sender repeatedly transfers σ_i to the receiver, using $\left(\frac{1}{nq''(n)}, neg(n), \frac{1}{nq''(n)}\right) - WOT$. The receiver determines the correct value using majority rule. The point is to decrease the probability that the receiver wrongly determines σ_i.

The Protocol

1. The sender transfers σ_i, n times to the receiver, using $\left(\frac{1}{nq''(n)}, neg(n), \frac{1}{nq''(n)}\right)-WOT$.
2. The receiver decides the value of σ_i by majority rule.

Analysis Sketch. The proof of the Correctness property is immediate by Chernoff bound. The Sender's privacy property is proved by hybrid argument and finally the proof of the Receiver's privacy property is proved by a simple information theoretic argument.

5.3 Using $(neg(n), neg(n), \frac{1}{3}) - WOT$ to construct $(neg(n), neg(n), neg(n)) - WOT$

In this final step, we reduced the third parameter into a negligible function. The implementation of this step follows the construction presented by Crépeau and Kilian [3].

5.4 Putting It All Together

Given a $\left(\frac{1}{q(n)}, 1 - \frac{1}{t(n)}, \frac{1}{q(n)}\right)$–$WOT$ protocol, where t is any positive polynomial and we are free to choose q as we like. We start by choosing $q(n)$ to be equal to $3n^2t(n)$. The second step is to use Step 5.1 to implement $\left(\frac{1}{3n}, neg(n), \frac{1}{3n}\right)$–$WOT$. In the third step we use Step 5.2 to implement $\left(neg(n), neg(n), \frac{1}{3}\right) - WOT$. In the last step we use Step 5.3 to implement the desired $(neg(n), neg(n), neg(n))$–WOT.

Recall that by the definition of $(\epsilon_1, \epsilon_2, \epsilon_3)$–$WOT$, $(neg(n), neg(n), neg(n))$–WOT is, in a sense, even a stronger protocol then OT. Since in OT all the requirements are computational knowledge ones and in $(neg(n), neg(n), neg(n))$–WOT the `Receiver`'s privacy property is negligible by information-theoretic means [9].

6 Further Issues

A natural question to ask is whether a similar result be obtained even if the permutation requirement is somewhat relaxed. For example can we construct an OT based on dense collection of injective one-way functions? The answer is positive when we consider length-preserving functions. Moreover, exactly the same construction as used in this text can be used. If the functions are not length-preserving, then the size of the function range must be dense both in 2^n and in 2^m (assuming that the function input is n bit long and the output is m bit long), and, again, exactly the same construction as used in this text can be used.

Another natural question to ask is whether an OT can be constructed using "standard" TDP, without any additional requirements? The answer seems to be negative, as it was proved by [8] that OT cannot be Black-Box reduced to collection of injective one-way functions and it seems likely, though not proven yet, that this result can be extended to TDP.

Acknowledgements. I would like to thank my MSc advisor, Oded Goldreich, who introduced me to this subject, and guided me through the writing of this paper.

[9] Note that this strengthening also happened in the EGL protocol.

References

1. Manuel Blum, *How to exchange (secret) keys*, ACM Transactions on Computer Systems **1** (1983), no. 2, 175–193.
2. G. Brassard, C. Crepeau, and J.-M. Robert, *Information theoretic reductions among disclosure problems*, 27th Annual Symp. on Foundations of Computer Science (FOCS '86) (Los Angeles, Ca., USA), IEEE, October 1986, pp. 168–173.
3. C. Crépeau and J. Kilian, *Weakening security assumptions and oblivious transfer*, Advances in Cryptology (CRYPTO '88) (Berlin - Heidelberg - New York), Springer, August 1990, pp. 2–7.
4. C. Crépeau and M. Sántha, *On the reversibility of oblivious transfer*, Proceedings of Advances in Cryptology (EUROCRYPT '91) (Berlin, Germany) (Donald W. Davies, ed.), LNCS, vol. 547, Springer, April 1991, pp. 106–113.
5. Claude Crépeau, *Equivalence between two flavours of oblivious transfers*, Advances in Cryptology—CRYPTO '87 (Carl Pomerance, ed.), Lecture Notes in Computer Science, vol. 293, Springer-Verlag, 1988, 16–20 August 1987, pp. 350–354.
6. Ivan Damgård, Joe Kilian, and Louis Salvail, *On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions*, Lecture Notes in Computer Science **1592** (1999), 56–??
7. S. Even, O. Goldreich, and A. Lempel, *A randomized protocol for signing contracts*, Communications of the ACM **28** (1985), no. 6, 637–647.
8. Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan, *The relationship between public key encryption and oblivious transfer*, 41st Annual Symp. on Foundations of Computer Science: proceedings: 12–14 November, 2000, Redondo Beach, California (IEEE, ed.), IEEE, 2000, pp. 325–335.
9. O. Goldreich and L. A. Levin, *A hard-core predicate for all one-way functions*, Proceedings of the twenty-first annual ACM Symp. on Theory of Computing, Seattle, Washington, May 15–17, 1989 (New York, NY, USA) (ACM, ed.), ACM Press, 1989, pp. 25–32.
10. O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game or A completeness theorem for protocols with honest majority*, Proc. 19th Ann. ACM Symp. on Theory of Computing, 1987, pp. 218–229.
11. Oded Goldreich, *Foundations of cryptography: Basic tools*, Cambridge University Press, Cambridge, UK, 2001.
12. ———, *Foundations of cryptography - volume 2*, Working Draft, available at www.wisdom.weizmann.ac.il/oded/foc-vol2.html, 2002.
13. Iftach Haitner, *Implementing oblivious transfer using collection of dense trapdoor permutations*, MSc thesis, available at www.wisdom.weizmann.ac.il/∼iftachh/papers/msc.html, 2003.
14. Joe Kilian, *Founding crytpography on oblivious transfer*, Proceedings of the 20th Annual ACM Symp. on the Theory of Computing (Chicago, IL) (Richard Cole, ed.), ACM Press, May 1988, pp. 20–31.
15. M. O. Rabin, *How to exchange secrets by oblivious transfer*, TR-81, Harvard, 1981.
16. A. De Santis and G. Persiano, *Zero-knowledge proofs of knowledge without interaction*, 33rd Annual Symp. on Foundations of Computer Science: October 24–27, 1992, Pittsburgh, Pennsylvania: proceedings [papers] (Silver Spring, MD 20910, USA) (IEEE, ed.), IEEE, 1992, pp. 427–436.
17. Andrew C. Yao, *How to generate and exchange secrets*, Proceedings of the 27th Symp. on Foundations of Computer Science (FOCS), IEEE Computer Society Press, 1986, pp. 162–167.