Implementing oblivious transfer using collection of dense trapdoor permutations

Iftach Haitner
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

January 6, 2004

Abstract

Until recently, the existence of collection of trapdoor permutations (TDP) was believed (and claimed) to imply oblivious transfer (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. The strengthening is in the "security" requirement of the TDP (i.e., the hardness to invert condition). Here we present an alternative construction that only requires the TDP to have "dense domains". Specifically we present an implementation of OT based on any dense TDP.

Contents

1 Introduction

1.1 Oblivious transfer (OT)

Oblivious transfer (OT), introduced by Rabin [Rab81], is a fundamental primitive in cryptography. OT has several equivalent formulations [Rab81, EGL85, CK90, Cré87, BCR86, CS91]. The version we studied, defined by Even, Goldreich and Lempel [EGL85], 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. At the end of the protocol, the `receiver` learns σ_i. In addition, the `sender` gains no knowledge about i and the `receiver` gains no knowledge about σ_{1-i}. (For details see Subsection 3.4).

OT implies key agreement (KA) [Rab81, Blu83], signing contracts [EGL85], and in general any secure multi-party evaluation [Yao86, GMW87].

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 (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 auxiliary efficient algorithms (in addition to the evaluation algorithm): 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. (For details see Subsection 3.6).

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 [Gol02]. This is due to the fact that in the standard TDP-based OT protocol, proposed by [EGL85], 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. As an example we note that the standard implementation of the Rabin's collection is not secured against an observer which gets, in addition to the element itself, the random coins used to generate the element [1]. 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 [2].

To enable the stronger security feature required by the OT, Goldreich [Gol02] 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. (For details see Subsection 3.7).

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 all strings of a particular length (for details see Subsection 3.8). The implications of this work are two-fold. First it might be easier to prove that a TDP has the density property rather than the enhancement assumed by [Gol02]. Second it might be a step towards implementing OT based merely on the existence of TDP.

We note that implementing OT based merely on the existence of TDP seems to be an hard task, since it was proved by [GKM$^+$00] that OT cannot

[1] Goldreich [Gol02] present an alternative implementation of the Rabin's collection, which is guaranteed to be secured against such an observer, assuming that the original implementation secured gainst an external observer (who only observes the sampled element).

[2] It is easy to see that the output of the "bad" sampler is uniformly distributed on the permutation's domain and therefore it is indeed a valid sampler. Moreover, given an element in the permutation domain along with the random coins used by the bad sampler to generate it, one can find out the pre-image of the element by evaluating the original sampler with the same random coins.

be *black-box* reduced to collection of injective trapdoor one-way functions and it seems likely, though not proven yet, that this result can be extended to TDP.

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 `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 selects 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).

We start by implementing a very weak form of OT, where we cannot assure that all the OT requirements (i.e., the secrecy of i and σ_{1-i}, and the correct computation of σ_i) hold, but we can guarantee that they hold with a noticeable probability. The implementation main idea is that the `sender` helps the `receiver` to check whether or not a given element is in the permutation domain, this help is done without delivering to the `receiver` "too much" information about the pre-image of the element.

Our implementation of a full-fledged OT follows by amplifying the above "weak OT". We note that few amplifications of information theoretic weak forms of OT (i.e., the OT secrecy requirements measure leak of information) are known (e.g., [CK90, DKS99]), but we are not aware of amplifications of computational knowledge weak forms of OT (i.e., at least one of the OT secrecy requirements measures leak of computational knowledge), such as our. Therefore, the amplification part of this thesis may be of independent interest.

1.6 The organization of the rest of the thesis

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 thesis. In Section 4 we present the high level design (the roadmap) of our construction. In Section 5 we give the full implementation of a weak form

OT based on dense-TDP, and in Sections 6-8 we show how to amplify such a “weak-OT” into a full-fledged one.

2 Overview of our construction

We present a polynomial time implementation of OT (oblivious transfer) based on the existence of dense-TDP (collection of trapdoor permutations). Our implementation follows the general ideas of the following OT protocol [EGL85].

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 (for details see Subsection 3.6). 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 (only) 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 domain sampler algorithm, D. In order to select r_i, the `receiver` first selects an 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. Recall that knowing the trapdoor t, the `sender` can invert 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 since 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 say that a given TDP has the "checkable-domains" property, if there is an efficient algorithm that checks whether an element is inside a given permutation domain (clearly, a given TDP might not have this property). We start by showing how to implement an OT based on dense-TDP (recall that a dense-TDP is a TDP whose permutations' domains are dense in $\{0,1\}^{k(n)}$ for some fixed positive polynomial k) having the checkable-domains property (hereafter referred to as c.d.d-TDP), and then step-by-step, show how to implement an OT using a standard dense-TDP. [3]

2.2.1 An OT based on c.d.d-TDP

We assume, without lost of generality, that the TDP permutations' domains are dense in $\{0,1\}^n$, rather than in $\{0,1\}^{k(n)}$. (A general construction for any fixed positive polynomial k is essentially the same).

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).

[3] We remark that any c.d.d-TDP can be transformed into a TDP whose permutations' domains are simply $\{0,1\}^{l(n)}$ for some fixed polynomial l. Hence, implementing OT can be achieve using the latter TDP with the standard [EGL85] protocol. We give the following implementation as we believe that it gives some intuition about the following steps.

c. r_i is set to $f_\alpha(s)$.

It is easy to see that the above construction is indeed an implementation of OT [4]. We stress that since the `receiver` did not use the collection sampler to selects r_{1-i}, the resulting protocol is secure 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 "trapdoor-c.d.d-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 trapdoor-c.d.d-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 full-fledged OT.

2.2.2 A weak OT based on trapdoor-c.d.d-TDP

The first idea is to try and use a similar protocol to the one in Subsection 2.2.1, 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 a 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 Subsection 2.2.1, otherwise the `receiver` restarts the protocol. It is easy to see, however, that this protocol leaks the value of i to the `sender` (because the `sender` gets both s and $r_i = f_\alpha(s)$)).

A better idea is for the `receiver` to send the `sender` $f_\alpha(s)$ [5] and r_{1-i} (instead of s and r_{1-i}) in a random order and the `sender` answers whether or not 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 Subsection 2.2.1. 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

[4] 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.

[5] By $f_\alpha(x)$ (resp. $f_\alpha^{-1}(x)$), where x is not guaranteed to be in D_α, we mean the result of invoking the collection evaluating algorithm, F (resp. F^{-1}), with inputs α and x (resp. α, x and t, where t is the trapdoor key of f_α). We stress that under this notation $f_\alpha^{-1}(x)$ is a single element in $\{0,1\}^n$ (i.e., $F^{-1}(\alpha,t,x)$) and not all the pre-images of x with respect to f_α.

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_α, 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_α is not a permutation on $\{0,1\}^n$, the values $f_\alpha(s)$ 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 a noticeable 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` some random information 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 r_0 or r_1 is not in D_α) it restarts the protocol. By keeping the amount of information the `sender` sends about pre-images small, we guarantee that only small amount of information about the $f_\alpha^{-1}(r_{1-i})$ (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 equal to $f_\alpha^{-1}(r_i)$. Hence, the protocol is a weak form of OT where all the required properties hold with noticeable probability.

In our implementation the random information that the `sender` sends to the `receiver` about the pre-images of r_0 and r_1 is the output of applying a randomly chosen pairwise independent hash function on the pre-images. The way we choose the parameters guarantees that only a small amount of information ($polylog(n)$ bits of information, where n is the security-parameter of the protocol) about the pre-images leaks to the `receiver`.

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).

2.2.3 A "very" weak OT based on any dense-TDP

The main idea is that any dense-TDP can be extended into a trapdoor-c.d.d-TDP. 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 $D_\alpha \subseteq D'_\alpha \stackrel{\text{def}}{=} \{x \in \{0,1\}^n \,|\, f_\alpha((f_\alpha^{-1}(x)) = x\}$ 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 noticeable (since $\frac{|D_\alpha|}{2^n}$ is noticeable), yet it may not be negligible close to 1 and so the extended collection's permutations are weak one-way permutations (rather then strong one-way). 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 extended (permutation) domain by checking whether or not $f_\alpha((f_\alpha^{-1}(x))$ is equal to x.

By using the protocol of Subsection 2.2.2 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 in Subsection 2.2.2 because 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 full-fledged OT.

2.2.4 The amplification step

The amplification of the above "very" weak OT into a full-fledged 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 full-fledged OT. The different parts of the amplification step are described through Sections 6 - 8.

3 Definitions

3.1 The semi-honest model

Loosely speaking, a semi-honest party (also known as an honest-but-curious 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 [GMW87] it is shown that semi-honest model protocols can be compiled into protocol for the general (malicious) model, in which nothing is assumed regarding the parties. (For details see [Gol02]). Having the above, we focus in this thesis on implementing OT in the semi-honest model.

3.2 The security-parameter of the protocol

The security-parameter of the protocol, denoted n, is given to the protocol in unary. Its purpose is to relate the time complexity of the protocol with its security features in the following way:

1. Determine the security quality of the protocol. The security features of the protocol are defined as functions of n.

2. Determine the time complexity of the protocol. The running time of a protocol is measured as a function of its input length. As the security-parameter length is n, a polynomial time protocol can only perform a polynomial in n number of steps. Note that for a protocol with fixed input length (e.g., OT), passing the security-parameter as one of its inputs is necessary in order to allow the protocol to perform a polynomial in n number of steps. By allowing the protocol to perform this number of steps, it can adapt itself to the required security features.

3.3 General notations

- negligible - We say that a function $\mu : \mathbb{N} \to [0, 1]$ is negligible in n ($neg(n)$), if for every positive polynomial p and all sufficiently large n's, it holds that $\mu(n) < \frac{1}{p(n)}$.

- noticeable- We say that a function $\mu : \mathbb{N} \to [0, 1]$ is noticeable in n, if there exists a positive polynomial p such that for every $n > 0$, it holds that $\mu(n) > \frac{1}{p(n)}$.

3.4 Oblivious transfer (OT)

Informally, a (one-out-of-two) Oblivious transfer 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. At the end of the protocol, the `receiver` learns σ_i. In addition, the `sender` gains no knowledge about i and the `receiver` gains no knowledge about σ_{1-i}. In this thesis, 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. Let us turn to the formal definition.

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. 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 $1 - neg(n)$, 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 [6]. Then for any probabilistic polynomial time algorithm M and for any choices of σ_i and i,

$$|Pr[M(VIEW_R(\sigma_i, 1, i)) = 1] - Pr[M(VIEW_R(\sigma_i, 0, i)) = 1]| = neg(n)$$

where the probability is over both parties' internal coin tosses.

3. `Receiver`'s privacy: The `sender` gains no computational knowledge about i. 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 probabilistic polynomial time algorithm M and for any choices of σ_i and σ_{1-i},

$$|Pr[M(VIEW_S(\sigma_i, \sigma_{1-i}, 1)) = 1] - Pr[M(VIEW_S(\sigma_i, \sigma_{1-i}, 0)) = 1]| = neg(n)$$

3.5 Weak OT: $(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$

Weak OT, parameterized as $(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$, is a two-party protocol that serves as an intermediate step in our implementation of OT. For $\epsilon_1, \epsilon_2, \epsilon_3 \geq 0$,

[6] 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.

$(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$ is the following 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 that 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. We usually omit the security-parameter from the protocol's input parameters list. $(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$ has the following properties:

1. Correctness: The `receiver` learns σ_i with probability at least $1-\epsilon_1$ (rather than 1 as in OT), where the probability is over both parties' internal coin tosses.

2. `Sender`'s privacy: The **computational knowledge** the `receiver` gains about σ_{1-i} is at most ϵ_2 (rather than negligible). More formally, for any probabilistic polynomial time algorithm M, for any choices of σ_i, i and large enough n,

$$|Pr[M(VIEW_R(\sigma_i, 1, i)) = 1] - Pr[M(VIEW_R(\sigma_i, 0, i)) = 1]| \leq \epsilon_2$$

   where $VIEW_R$ is defined in Subsection 3.4 and the probability is over both parties' internal coin tosses.

3. `Receiver`'s privacy: The **information** the `sender` gain about i is at most ϵ_3 (rather than negligible). More formally, for any choices of σ_i and σ_{1-i} and large enough n,

$$stat(VIEW_S(\sigma_i, \sigma_{1-i}, 1), VIEW_S(\sigma_i, \sigma_{1-i}, 0)) \leq \epsilon_3$$

   where $VIEW_S$ is defined in Subsection 3.4 and $stat$ stands for the statistical difference.

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.

3.6 Collection of trapdoor permutations (TDP)

Collection of trapdoor permutations (TDP) is a special case of collection of one-way permutations (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 auxiliary efficient algorithms (in addition to the evaluation algorithm): 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. Let us turn to the formal definition.

Definition: Collection of trapdoor permutations (uniform complexity version) [Gol01]: 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. Permutation sampler: $Pr[I(1^n) \in \overline{I}_n \times \{0,1\}^*] > 1 - 2^{-n}$.

   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, with out lost of generality, $D_i \subseteq \{0,1\}^{poly(|i|)}$.

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_i, algorithm 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 $X_n \stackrel{\text{def}}{=} D(I_n)$, then for any probabilistic polynomial time algorithm M, every positive 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 $x \in D_i$, $Pr[F^{-1}(i,t,f_i(x)) = x] > 1 - 2^{-n}$.

   That is, given a permutation along with its trapdoor, F^{-1} is used to find the pre-image of any element in its domain.

In this thesis we use an alternative, though equivalent, version of the fourth condition, where we replace (the random variable) X_n of conditioned 4, by the equivalent random variable $f_{I_n}^{-1}(X_n)$. Hence we have the following condition:

4′. $Pr[M(I_n, X_n) = f_{I_n}^{-1}(X_n)] < \frac{1}{p(n)}$.

3.7 Enhanced collection of trapdoor permutations

The implementation of OT presented by [EGL85], is based on the existence of enhanced TDP. The enhancement refers to the hard-to-invert condition (i.e., that 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 [Gol02]).

Formally, the enhanced definition has the form: Let (I, D, F) be a TDP, and let $\overline{D}$ be the deterministic version of the permutation sampler D, in which the algorithm receives, as an additional input, its random coins input; that is $D(i) = \overline{D}(i,r)$ for r uniformly distributed in $\{0,1\}^{poly(|i|)}$. Let I_n be the random variable describing the distribution of the permutation indices in the collection. Then in the enhanced TDP, the hard-to-invert condition has the form: For any probabilistic polynomial time algorithm M and every positive polynomial p,

$$Pr[M(I_n, r) = f_{I_n}^{-1}(\overline{D}(I_n, r))] < \frac{1}{p(n)}$$

That is, the inverting algorithm M receives the random coins r rather than only the corresponding image $\overline{D}(i, r)$.

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

3.8 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, recall that D_α is the domain of the permutation named α. The additional requirement is that 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 essentially equivalent.

4 Our implementation roadmap

Our implementation consists of four major steps. In the first step we construct a very weak protocol and then step-by-step we amplify the protocol till we achieve a full-fledged OT. Each of the amplification steps (the last three steps), however, can be used independently in order to amplify a weak form of OT into a stronger one. In particular, by combing the three amplification steps together, one can amplify any $\left(\frac{1}{3n^2 t(n)}, 1 - \frac{1}{t(n)}, \frac{1}{3n^2 t(n)}\right) - WOT$ (where t is any positive polynomial) into an OT. Following is a short description of the implementation's steps.

1. Using any dense-TDP, we construct a $\left(\frac{1}{q(n)}, 1-\frac{\rho(n)^2}{4}, \frac{1}{q(n)}\right)-WOT$, where ρ is the density parameter of the collection and q is any positive polynomial.

   In this step we implement a very weak form of $(\epsilon_1,\epsilon_2,\epsilon_3)-WOT$, where 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 in the next step in order to construct a stronger protocol.

2. Using any $\left(\frac{1}{nq'(n)t(n)}, 1-\frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)$–$WOT$, we construct a $\left(\frac{1}{q'(n)}, neg(n), \frac{1}{q'(n)}\right)$–$WOT$, where q' and t are any positive polynomials.

   In this step, we show how to reduce (the potentially big) second parameter of the given WOT into a negligible function. Note that the first and third parameters increase by a factor of $nt(n)$.

   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 $\left(\frac{1}{nq'(n)t(n)}, 1-\frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)$–$WOT$. 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.

3. Using any $\left(\frac{1}{nq''(n)}, neg(n), \frac{1}{nq''(n)}\right)$–$WOT$, we construct a $\left(neg(n), neg(n), \frac{1}{q''(n)}\right)$–$WOT$, where q'' is any positive polynomial.

   In this step, we show how to reduce the first parameter into a negligible function. Note that the third parameter increases by a factor of n.

   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. Thus the probability that the `receiver` did not get σ_i correctly, vanishes exponentially.

4. Using any $(neg(n), neg(n), \frac{1}{3})$–$WOT$, we 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 [CK90].

Thus in order to implement OT, we do the following steps: First, we implement, using dense-TDP, a $\left(\frac{\rho(n)^2}{12n^2}, 1-\frac{\rho(n)^2}{4}, \frac{\rho(n)^2}{12n^2}\right)-WOT$. Next, we use the latter protocol to construct $\left(\frac{1}{3n}, neg(n), \frac{1}{3n}\right)-WOT$, which in turn is used to construct $\left(neg(n), neg(n), \frac{1}{3}\right)-WOT$, which is finally used to construct the desired $(neg(n), neg(n), neg(n))-WOT$.

Recall that by the definition of $(\epsilon_1, \epsilon_2, \epsilon_3)$–$WOT$, the constructed $(neg(n), neg(n), neg(n))$–WOT is actually stronger protocol than OT. This is the case since in OT all the requirements refer to computational knowledge, whereas in $(neg(n), neg(n), neg(n))$–WOT the `Receiver`'s privacy property is information-theoretic. (This strengthening also occurs in the EGL protocol).

5 Using dense-TDP to construct $\left(\frac{1}{q(n)}, 1-\frac{\rho(n)^2}{4}, \frac{1}{q(n)}\right)$–$WOT$

Recall that ρ is the density parameter of the collection and q is any positive polynomial.

In this section we implement a very weak form of $(\epsilon_1, \epsilon_2, \epsilon_3)-WOT$, where 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 in the next section in order to construct a stronger protocol.

5.1 Preliminaries

Let (I, D, F) be a dense-TDP with density parameter ρ. For simplicity's sake, we assume that the evaluation and inverting algorithms (i.e., F and F^{-1}) are deterministic and errorless; that is always return the right answers. Note that in the definition of dense-TDP, all the collection's algorithms are probabilistic and might return wrong answers with negligible probability. The extension of the following implementation to the general case is done in Subsection 5.5.

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 without lost of generality 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. Note that under the extended notation f_α is no longer guaranteed to be a permutation. We stress, however, that under the extended notation $f_\alpha^{-1}(x)$ denotes a single element in $\{0,1\}^n$ (i.e., $F^{-1}(\alpha,t,x)$) and not all the pre-images of x with respect to f_α.

5.2 The protocol's outline

Our 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 $\{0,1\}^n$. (We stress that unlike in the EGL protocol, here the domain sampler algorithm (D) is not used at all). Note that either r_{1-i} or s might not be in D_α.
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 equal to $f_\alpha^{-1}(r_i)$. The above help is given to the `receiver` without leaking "too much" information about the value of r_{1-i} (and hence about the value of σ_{1-i}). This help is needed, as there is no efficient way to the `receiver` to tell 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 equal to $f_\alpha^{-1}(r_i)$, 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.

5.3 The protocol itself

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

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. However, in case that the protocol is not working correctly, the protocol detects it, with sufficiently high probability, in Steps 4 - 5 and restarts.

3. The `receiver` sends (r_0, r_1) to the `sender` in a 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 a 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 select 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's distribution. For example, it might be that for all x not in D_α, the value of $f_\alpha(x)$ is equal to 0^n and hence with high probability (i.e., $1-\rho(n)$) $r_i = 0^n$. Indeed in such a case and if r_0 and r_1 would have been sent in a fixed order, the `receiver` could guess the value of i by checking whether or not (for $j = 0, 1$) $r_j = 0^n$ (note that r_{1-i} is always uniformly distributed in $\{0,1\}^n$).

4. For both $j = 0, 1$, the `sender` checks whether or not $f_\alpha(f_\alpha^{-1}(w_j)) = w_j$. If both answers are 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`.

   That is, the `sender` does not abort the current iteration, only if both r_0 and r_1 are in $D'_\alpha \stackrel{\text{def}}{=} \{x \in \{0,1\}^n \,|f_\alpha((f_\alpha^{-1}(x)) = x\}$. If the current iteration is not aborted, some random information about $f_\alpha^{-1}(r_0)$ and $f_\alpha^{-1}(r_1)$ is delivered to the `receiver`. The amount of this information, however, is small and thus does not enable the `receiver` to compute $f_\alpha^{-1}(r_{1-i})$.

5. The `receiver` **aborts** the current iteration if $v_{i\oplus k} \neq h(s)$.

   Motivation: 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 $s = f_\alpha^{-1}(r_i)$ and the second requirement is that r_i and r_{1-i} "look" as though they have been chosen from the same distribution. By Step 4 we are guaranteed that both r_0 and r_1 are in D'_α. Therefore, we are guaranteed that r_{1-i} is uniformly distributed in D'_α (as r_{1-i} was uniformly chosen in $\{0,1\}^n$). We are not guaranteed, however, that this is the case with r_{1-i}. The probability that the value of r_i is equal to a given element in D'_α, is determined by the number of pre-images (with respect to f_α) of this element in $\{0,1\}^n$ and this number might not be the same for different elements in D'_α (e.g., it might be the case that for all x not in D'_α, the value of $f_\alpha(x) = c$, where c is a fixed element in D'_α). We are guaranteed, however, that if $f_\alpha(r_i) = s$ then r_i is uniformly distributed in D'_α (by a similar argument to the one we gave about the distribution of r_{1-i}). 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 $f_\alpha(r_i) = s$ and hence the above two requirements are guaranteed to hold. Moreover, in such a case 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 high probability (i.e., at least $1-\frac{1}{q(n)}$).

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 selects $y_j \in \{0,1\}^n$ 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 $g_\alpha(x,y) \stackrel{\text{def}}{=} (f_\alpha(x), y)$. In contrast, 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_α. 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 $f_\alpha^{-1}(r_i) = s$, 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`.

5.4 Analysis

In the following analysis we refers to the above protocol as *the protocol.*

We start by proving that the protocol's running time is polynomial, and then we prove that the protocol is a $\left(\frac{1}{q(n)}, 1-\frac{\rho(n)^2}{4}, \frac{1}{q(n)}\right)$–$WOT$. The latter is done by first proving the Correctness and the `Receiver`'s privacy properties of the protocol, by analyzing the probability that the protocol ends in a certain way that guarantees the above properties. Then we, separately, prove the `Sender`'s privacy property.

5.4.1 The running time of the protocol

By the density property of the collection we have that each iteration has probability of $\frac{1}{\rho(n)^2}$ to be the last one (because when both r_0 and r_1 are in D_α, the protocol is guaranteed to halt). Therefore, with very high probability, the protocol halts after $\frac{n}{\rho(n)^2}$ iterations. Moreover, the protocol can be extended such that it always halts after $\frac{n}{\rho(n)^2}$ iterations. This modification can be ignored while analyzing the protocol, as it only effects the protocol behavior with negligible probability.

5.4.2 The chances for a "good-ending"

Note that the part of the protocol prior to its last iteration does not leak any information to the `sender` or the `receiver`, because this part (which contains the iterations that were aborted before Step 6) is, essentially, independent of σ_0, σ_1 and i. Having the above, we focus in analyzing the protocol's last iteration.

We say that the protocol had a good-ending, if in its last iteration $s = f_\alpha^{-1}(r_i)$, otherwise we say that the protocol had a bad-ending. The proof of the protocol's Correctness and `Receiver`'s privacy properties stems from the following claim about the probability that the protocol had a good-ending.

Claim 5.4.1 *The probability over the random coins used by the protocol that the protocol had a good-ending is at least* $1 - \frac{1}{q(n)}$.

Proof: Let's say that the protocol had a t-good-ending if the protocol had exactly t iterations and a good-ending. Similarly, we define the notation of t-bad-ending. Let's denote by γ the conditional probability of having t-good-ending given that the protocol had *at least* t iterations. Note that, by the protocol structure, γ is indeed independent of t. In the same manner we denote by β the conditional probability of having t-bad-ending given that the protocol had at least t iterations (note that β is independent of t as well). We note that clearly $\beta + \gamma \leq 1$ and when the protocol is not guaranteed to have only one iteration, we have that $\beta + \gamma < 1$.

Using the above notation we are ready to analyze the probability for a bad-ending. If the protocol had a bad-ending, then there exists a positive t such that the protocol had a t-bad-ending and for any j smaller than t the protocol did not end, and in particular did not have a j-good-ending. Therefore the probability that the protocol had a bad-ending is smaller than $\sum_{t\geq 0}(1-\gamma)^t\beta = \frac{\beta}{\gamma}$.

By the density property of the collection we have that $\gamma \geq \rho^2(n)$ (as if in the t iteration, both r_0 and r_1 are in D_α, then the protocol is guaranteed to have t-good-ending).

In order to have a t-bad-ending (conditioned on the protocol had at least t iteration), it should be the case that in the t'th iteration $s \neq f_\alpha^{-1}(r_i)$ but still both s and $f_\alpha^{-1}(r_i)$ have the same hash-value, with respect to the uniformly chosen pairwise hash function. As the range of the pairwise hash function we use in the protocol is $\frac{q^2(n)}{\rho(n)}$, we have that the probability to have a t-bad-ending is bounded above by $\frac{\rho(n)}{q^2(n)}$. Therefore the probability of a bad-ending is not more than $\frac{\left(\frac{\rho(n)^2}{q(n)}\right)}{\rho^2(n)} = \frac{1}{q(n)}$. ∎

Having established the above claim we can prove the protocol's Correctness and `Receiver`'s privacy properties.

5.4.3 Proving the Correctness property - The `receiver` outputs σ_i with probability greater than $1 - \frac{1}{q(n)}$

If the protocol had a good-ending (and therefore $s = f_\alpha^{-1}(r_i)$), then the bit the `receiver` computes as the hardcore bit of (r_i, y_i) (i.e., $b(s, y_i)$) is the right one. Therefore, if the protocol had a good-ending, then the `receiver` outputs σ_i, and by Claim 5.4.1, it happens with probability at least $1 - \frac{1}{q(n)}$.

5.4.4 Proving the `Receiver`'s privacy property - The `sender` does not gain more information about i than $\frac{1}{q(n)}$

The crucial observation is that given that the protocol had a good-ending, $VIEW_S(\sigma_0, \sigma_1, 0)$ and that $VIEW_S(\sigma_0, \sigma_1, 1)$ have exactly the same distribution. To prove the above, let's recall the definition of the set D'_α (i.e., $D'_\alpha = \{x \in \{0,1\}^n \, | f_\alpha(f_\alpha^{-1}(x)) = x\}$). As the last iteration was not aborted, we have that r_{1-i} is uniformly distributed in D'_α. The latter is true as r_{1-i} was uniformly chosen in $\{0,1\}^n$, and due to Step 4 it holds that r_{1-i} is in D'_α. Given that the protocol had a good-ending (and thus $s = f_\alpha^{-1}(r_i)$), s is uniformly distributed in $f_\alpha^{-1}(D'_\alpha) \stackrel{\text{def}}{=} \{f_\alpha^{-1}(x) | x \in D'_\alpha\}$ (by a similar argument to the one we gave about the distribution of r_{1-i}). Hence (as f_α is clearly one-to-one and onto D'_α), r_i is uniformly distributed in D'_α. Therefore the statistical difference between $VIEW_S(\sigma_0, \sigma_1, 1)$ and $VIEW_S(\sigma_0, \sigma_1, 0)$ is bounded above by the probability that the protocol had a bad-ending, and therefore by $\frac{1}{q(n)}$.

5.4.5 Proving the `Sender`'s privacy property - The `receiver` does not gain more computational knowledge about σ_{1-i} than $1-\frac{\rho(n)^2}{4}$

Recall that the above condition states that the `receiver` has at least $\frac{\rho(n)^2}{4}$ uncertainty regarding the value of σ_{1-i}. We show that it is impossible for the `receiver` to gain more computational knowledge about σ_{1-i} than the above limit. Intuitively, if the `receiver` does gain more knowledge, then it has to be "significantly" successful doing so, in the specific case when in the last iteration both s and r_{1-i} are in D_α. The reason is that by the density property of the collection, the probability that in the *last* iteration, both s and r_{1-i} happen to be in D_α, is at least $\frac{1}{\rho^2}$. Hence, if the `receiver` gains more computational knowledge about σ_{1-i} than $1-\frac{\rho(n)^2}{4}$, then the `receiver` gains at least $\frac{3}{4}$ computational knowledge about σ_{1-i}, conditioned on both s and r_{1-i} being in D_α. Gaining such a knowledge about σ_{1-i} under the latter conditioning is impossible, as it leads to inverting the collection of one-way permutations itself.

We remark that gaining $1-\rho(n)$ computational knowledge about σ_{1-i} might be easy. For example, it might be the case that outside D_α, the function f_α is easy to invert (e.g., f_α might be the identity function for any x not in D_α) and the probability that r_{1-i} is not in D_α might be as high as $1-\rho(n)$. Hence, one could succeed in computing $f_\alpha^{-1}(r_{1-i})$ with probability $1-\rho(n)$ and thus could compute σ_{1-i} with probability $1-\rho(n)$. Recall that we claimed that computing σ_{1-i} with probability greater than $1-\frac{\rho(n)^2}{4}$, is at least as hard as inverting the collection itself and therefore impossible. Thus there is a gap between the stated lower and upper bound on the amount of computational knowledge the `receiver` can gain about σ_{1-i} (their values are $1-\rho(n)$ and $1-\frac{\rho(n)^2}{4}$ respectively).

Let us turn to the formal proof. We assume, by contradiction, that the above condition does not hold (i.e., the `receiver` does gain more computational knowledge about σ_{1-i} than $1-\frac{\rho(n)^2}{4}$) and construct a polynomial time algorithm that inverts the dense-TDP with a non-negligible success. The proof has two major steps. First we construct a polynomial time algorithm, B, that computes $b(f_\alpha^{-1}(x), y)$ with non-negligible probability. (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, A, that computes ,with non-negligible probability, $f_\alpha^{-1}(x)$, by embedding B in the reduction given by [GL89], which establishes that b is an hard core predicate for any one-way function.

First step - constructing Algorithm B

We present a polynomial time algorithm B that (under the contradiction assumption) computes $b(f_\alpha^{-1}(x),y)$ with non-negligible probability. It has five inputs: α, h, v, x and y. Assuming that x is uniformly distributed in D_α, y is uniformly distributed in $\{0,1\}^n$ and that its other inputs were "properly chosen" (to be define below), B computes the value of $b(f_\alpha^{-1}(x),y)$ with non-negligible probability. It does so by creating a random looking `receiver`'s view of the protocol, in a way that knowing the value of σ_{1-i} yields the value of $b(f_\alpha^{-1}(x),y)$. Let us turn to the construction of B.

$B(\alpha,h,v,x,y)$: Recall that under the contradiction assumption there exist an algorithm M, an index i and a secret σ_i such that for infinitely many n's

$$|Pr[M(VIEW_R(\sigma_i,1,i))=1] - Pr[M(VIEW_R(\sigma_i,0,i))=1]| > \frac{\rho(n)^2}{4}$$

Algorithm B consists on the following steps:

1. Generate $V^{\alpha,h,v,x,y}$, to be defined below.

   Under some conditions (detailed below) the random variable $V^{\alpha,h,v,x,y}$ would have the same distribution as the `receiver`'s view of the protocol. Moreover, the generation would guarantee that knowing the value of σ_{1-i} in $V^{\alpha,h,v,x,y}$ yields the value of $b(f_\alpha^{-1}(x),y)$. The generation is done through the following steps.

   (a) Generate the first part of $V^{\alpha,h,v,x,y}$ - the part prior to the last iteration.

   Recall that the `receiver`'s view of the protocol is a concatenating of several (one to polynomial many) transcripts of a single iteration. Also recall that, except for the last iteration, all iterations are independent of the values of σ_0 and σ_1. The generation of this part is done by simulating the protocol with the `sender`'s secrets set to one (any fixed values would do) and the `receiver`'s index set to i. After the simulation ends, the last iteration is removed.

Motivation: If the `receiver`'s index in the protocol is i, then the first part of $V^{\alpha,h,v,x,y}$ has the same distribution as the first part of the `receiver`'s view of the protocol.

(b) Generate the second part of $V^{\alpha,h,v,x,y}$ - the last iteration part.

- Set the permutation and hash function of this iteration to the inputs parameters α and h respectively.
- Choose s uniformly in D_α, y_i uniformly in $\{0,1\}^n$ and set r_i to $f_\alpha(s)$, v_i to $h(s)$ and c_i to $b(s, y_i) \oplus \sigma_i$.
  Motivation: If in the protocol's last iteration the permutation is α, the hash function is h and s is in D_α, then the values of the i'th-subscripted variables (e.g., y_i) and the value of s in $V^{\alpha,h,v,x,y}$'s last iteration, have the same distribution as these values in (the last iteration of) the `receiver`'s view of the protocol.
- Set r_{1-i} to x, y_{1-i} to y and v_{1-i} to v, and select c_{1-i} uniformly in $\{0,1\}$.
  Motivation: If x is uniformly distributed in D_α, y is uniformly distributed in $\{0,1\}^n$, $v = h(f_\alpha^{-1}(x))$ and σ_{1-i} is uniformly distributed in $\{0,1\}$, then the remaining values of $V^{\alpha,h,v,x,y}$'s last iteration (i.e., r_{1-i}, y_{1-i}, v_{1-i} and c_{1-i}) have the same distribution as these values in (the last iteration of) the `receiver`'s view of the protocol.

2. Return $M(V^{\alpha,h,v,x,y}) \oplus c_{1-i}$ as the value of $b(f_\alpha^{-1}(x), y)$.

We now prove an essential claim about the probability that Algorithm B succeeds in computing $b(f_\alpha(x), y)$.

Claim 5.4.2 *Assuming that α, x, h and y are uniformly distributed in I_n, D_α, H_n and $\{0,1\}^n$ respectively and that $v = h(f_\alpha^{-1}(x))$, then $B(\alpha, h, v, x, y) = b(f_\alpha^{-1}(x), y)$ with probability at least $\frac{3}{4}$.*

Proof: First, let's recall that by the contradiction assumption, we have that for infinitely many n's,

$$|Pr[M(VIEW_R(\sigma_i, 1, i)) = 1] - Pr[M(VIEW_R(\sigma_i, 0, i)) = 1]| > 1 - \frac{\rho(n)^2}{4}$$

Without loss of generality we can drop off the absolute value notation, therefore we have that the first item (i.e., $Pr[M(VIEW_R(\sigma_i, 1, i)) = 1]$) is not smaller than $1-\frac{\rho(n)^2}{4}$ and that the second item (i.e., $Pr[M(VIEW_R(\sigma_i, 0, i)) = 1]$) is not bigger than $\frac{\rho(n)^2}{4}$. Thus we have that for both $\sigma_{1-i} = 0, 1$

$$Pr[M(VIEW_R(\sigma_i, \sigma_{1-i}, i)) = \sigma_{1-i}] > 1 - \frac{\rho(n)^2}{4} \tag{1}$$

By the density property of the collection, we have that the (a priori) probability, over the random coins used by the protocol, that in the last iteration both s and r_{1-i} are in D_α, is at least $\frac{1}{\rho^2}$. Hence, by Equation 1 and a simple average argument, we have that for both $\sigma_{1-i} = 0, 1$

$$Pr[M(VIEW_R(\sigma_i, \sigma_{1-i}, i)) = \sigma_{1-i} | s, r_{1-i} \in D_\alpha] > \frac{3}{4} \tag{2}$$

We are about to show that assuming that α, x, h and y are uniformly distributed in I_n, D_α, H_n and $\{0,1\}^n$ respectively and that $v = h(f_\alpha^{-1}(x))$, the emulated view generated by Algorithm B (i.e., $V^{\alpha,h,v,x,y}$) has the same distribution as the `receiver`'s view of the protocol, conditioned on i being the `receiver`'s index, σ_i being the `sender`'s i indexed secret and s and r_{1-i} being in D_α. (Therefore, by Equation 2, invoking M on $V^{\alpha,h,v,x,y}$ yields, with probability at least $\frac{3}{4}$, the value of σ_{1-i} and hence, Algorithm B returns $b(f_\alpha^{-1}(x), y)$ with probability at least $\frac{3}{4}$).

For a given value of $V^{\alpha,h,v,x,y}$, we define $\tau(V^{\alpha,h,v,x,y})$ as $b(f_\alpha^{-1}(x), y) \oplus c_{1-i}$, where x, y and c_{1-i} are the values of these variables in $V^{\alpha,h,v,x,y}$. Let $V_0^{\alpha,h,v,x,y}$ be the random variable defined by the distribution of $V^{\alpha,h,v,x,y}$ conditioned on $\tau(V^{\alpha,h,v,x,y}) = 0$ and similarly we define $V_1^{\alpha,h,v,x,y}$. By the construction of $V^{\alpha,h,v,x,y}$, the assumption that α, h, x and y are uniformly distributed in I_n, H_n, D_α and $\{0,1\}^n$ respectively and the assumption that $v = h(f_\alpha^{-1}(x))$, we have that for both σ_{1-i} in $\{0,1\}$, $V_{\sigma_{1-i}}^{\alpha,h,v,x,y}$ has the same distribution as $VIEW_R(\sigma_i, \sigma_{1-i}, i)$, conditioned on s and r_{1-i} being in D_α. Hence,

$$\begin{aligned} &Pr[M(V^{\alpha,h,v,x,y}) = \tau(V^{\alpha,h,v,x,y})] = \\ &= Pr[M(VIEW_R(\sigma_i, \tau(V^{\alpha,h,v,x,y}), i)) = \tau(V^{\alpha,h,v,x,y}) | s, r_{1-i} \in D_\alpha] \end{aligned} \tag{3}$$

where α, x, h and y are uniformly chosen from I_n, D_α, H_n and $\{0,1\}^n$ respectively.

Now we are ready to prove the claim. By Step 2 of the algorithm and the definition of τ we have that

$$\begin{aligned}
& Pr[B(\alpha,h,h(x),x,y) = b(f_\alpha^{-1}(x),y)] \\
&= Pr[M(V^{\alpha,h,v,x,y}) \oplus c_{1-i} = b(f_\alpha^{-1}(x),y)] \\
&= Pr[M(V^{\alpha,h,v,x,y}) = \tau(V^{\alpha,h,v,x,y})]
\end{aligned}$$

where α, x, h and y are uniformly chosen from I_n, D_α, H_n and $\{0,1\}^n$ respectively. Therefore by the above result and Equations 2 and 3, we have proved that:

$$Pr[B(\alpha,h,h(x),x,y) = b(f_\alpha^{-1}(x),y)] \geq \frac{3}{4}$$

where α, x, h and y are uniformly chosen from I_n, D_α, H_n and $\{0,1\}^n$ respectively. ∎

Second step - constructing Algorithm A

We are about to use Algorithm B, in order to construct a polynomial time algorithm that finds, with non-negligible success, $f_\alpha^{-1}(x)$. Recall that $b(z,y)$ is the inner product of z and y mod 2. We call a random process b^z a predictor for $b(z,\cdot)$, if

$$Pr_{y\in_R\{0,1\}^n}[b^z(y) = b(z,y)] > \frac{1}{2} + \epsilon$$

where ϵ is some fixed positive constant, and the probability is taken uniformly over the internal coins tosses of b^z and all possible choices of $y \in \{0,1\}^n$. By [Gol99, Thm. C4] variant of the reduction used by [GL89] in proving *hard-core predicate for any one-way function* [7], we have that one can use a predictor for $b(z,\cdot)$ to find, with non-negligible success, the value of z. The following algorithm uses Algorithm B to generate a predictor for $b(f_\alpha^{-1}(x),\cdot)$, and then use the above reduction to compute the value of $f_\alpha^{-1}(x)$. Let us turn to the formal definition.

$A(\alpha, x)$:

1. Select h uniformly in H_n, and v uniformly in R_h (where R_h stands for h range).

[7] The variant presented in [Gol99, Thm. C4], was originally suggested by Charlie Rackoff.

2. Try to generate a predictor for $b(f_\alpha^{-1}(x),\cdot)$: Generate $P^{\alpha,h,v,x}$, where $P^{\alpha,h,v,x}$ is a one input algorithm, constructed from Algorithm B by hardwiring B's first four inputs to α, h, v and x, where α and x are the current algorithm inputs, and h and v were chosen above.

   Motivation: With non-negligible probability (to be analyzed below), $P^{\alpha,h,v,x}$ is a predictor for $b(f_\alpha^{-1}(x),\cdot)$

3. Compute $f_\alpha^{-1}(x)$ using the reduction given by [Gol99, Thm. C4] in proving hard-core predicate for any one-way function, i.e., the algorithm uses $P^{\alpha,h,v,x}$ as the random process used in the reduction to compute $f_\alpha^{-1}(x)$.

Lemma 5.4.3 *A inverts the collection with non-negligible probability.*

Proof:

The proof is an immediate result of the following claim concerning the probability that $P^{\alpha,h,v,x}$ is a predictor for $b(f_\alpha^{-1}(x),\cdot)$:

Claim 5.4.4 *Assuming that α and x are uniformly distributed in I_n and D_α respectively, then, with non-negligible probability, $P^{\alpha,h,v,x}$ is a predictor for $b(f_\alpha^{-1}(x),\cdot)$.*

Proof: By Claim 5.4.2

$$Pr[B(\alpha,h,h(f_\alpha^{-1}(x)),x,y) = b(f_\alpha^{-1}(x),y)] \geq \frac{3}{4}$$

where α, h, x and y are uniformly chosen from I_n, H_n, D_α and $\{0,1\}^n$ respectively. By a simple averaging argument, we have that for a constant fraction of the triplets $\{(\alpha,h,x)\}$, where $\alpha \in I_n$, $h \in H_n$ and $x \in D_\alpha$, it holds that

$$Pr_{y\in_R\{0,1\}^n}[B(\alpha,h,h(f_\alpha^{-1}(x)),x,y) = b(f_\alpha^{-1}(x),y)] \geq \frac{17}{32} \qquad (4)$$

By Algorithm A Step 2, we have that if $v = h(f_\alpha^{-1}(x))$, then $B(\alpha,h,h(f_\alpha^{-1}(x)),x,y) = P^{\alpha,h,v,x}(y)$. Note that because v was uniformly chosen in R_h, the probability that $v = h(f_\alpha^{-1}(x))$ is noticeable (recall that $|R_h| = \frac{q(n)}{\rho^2(n)}$). Thus, by the above and Equation 4, the proof follows.

▮

The lemma follows by combining Claim 5.4.4 and [Gol99, Thm. C4]. ▮

5.5 Extending the protocol to the general version of dense-TDP (all the collection's algorithms might have errors)

In the general version of dense-TDP, all the collection's algorithms (i.e., I, D, F and F^{-1}) are probabilistic and might, though with negligible probability, have errors. Recall that in the above construction we assumed that F and F^{-1} are deterministic and errorless. We would like now to present a slightly different version of the previous protocol that is based on general version of dense-TDP and is still a $\left(\frac{1}{q(n)}, 1-\frac{\rho(n)^2}{4}, \frac{1}{q(n)}\right)-WOT$.

Let $\overline{F}$ and $\overline{F}^{-1}$ be the deterministic versions of the F and F^{-1} that receive their random coins as an additional input. The modified version of Protocol 5.3 has the following modifications: In the first line of the protocol (Step 1) the `sender` selects, in addition to its other choices, two random strings z_1, z_2 and sends z_1 to the `receiver`. Whenever each of the parities evaluates $F(\alpha, x)$ or $F^{-1}(\alpha, x, t)$, (where t is the trapdoor of α) it does so by evaluating $\overline{F}(\alpha, x, z_1)$ or $\overline{F}^{-1}(\alpha, x, t, z_2)$ respectively.

In the analysis of the original protocol we implicitly assumed that for any x in D_α, $F(\alpha, x)$ and $F^{-1}(\alpha, x, t)$ always return $f_\alpha(x)$ and $f_\alpha^{-1}(x)$ respectively. In the above modified version of the protocol these assumption are only guaranteed to hold with very high probability (i.e., $1-2^{-n}$). Nevertheless, the same proof of correctness may be used to prove that the modified version is still a $\left(\frac{1}{q(n)}, 1-\frac{\rho(n)^2}{4}, \frac{1}{q(n)}\right)-WOT$.

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

Recall that q' and t are any positive polynomials.

In the following protocol, the `sender` splits its original pair of secrets into many pairs of secrets, by splitting each of the original secrets into many secrets (i.e., $nt(n)$) 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 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.

6.1 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 $k = 0, 1$, the `sender` sets the following values:

   - $\omega_{k,1}, \ldots, \omega_{k,nt(n)-1}$ are uniformly chosen at $\{0, 1\}$.
   - $\omega_{k,nt(n)}$ is set to $(\bigoplus_{j=1}^{nt(n)-1} \omega_{k,j}) \oplus \sigma_k$.

   Motivation: The `sender` splits each of its original secrets into $nt(n)$ shares called ω's such that $\bigoplus_{j=1}^{nt(n)} \omega_{k,j} = \sigma_k$. Thus, in order to know σ_k one should know $\omega_{k,1}, \ldots, \omega_{k,nt(n)}$.

2. For all $1 \leq j \leq nt(n)$, the `sender` transfers $\omega_{i,j}$ to the `receiver`, using $\left(\frac{1}{nq'(n)t(n)}, 1 - \frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)-WOT$.

   That is, the parties invoke $\left(\frac{1}{nq'(n)t(n)}, 1 - \frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)$–$WOT$ for $nt(n)$ times such that in the j'th invoking the `sender`'s secrets are $\omega_{0,j}$ and $\omega_{1,j}$ and the `receiver`'s index is i.

3. The `receiver` locally outputs $\bigoplus_{j=1}^{nt(n)} \omega_{i,j}$.

6.2 Analysis

In the following analysis we refer to the above protocol as the *high-level protocol*, and we refer to $\left(\frac{1}{nq'(n)t(n)}, 1 - \frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)-WOT$ as the *subprotocol*.

6.2.1 Proving the Correctness property - The `receiver` outputs σ_i with probability greater than $1 - \frac{1}{q'(n)}$

By the Correctness property of the subprotocol, for every j the `receiver` obtain (in the high-level protocol) the value of $\omega_{j,i}$ with probability at least $1 - \frac{1}{nq'(n)t(n)}$. Therefore, by union bound, the probability that the `receiver` obtains values of all $\omega_{j,i}$ correctly (and hence, outputs σ_i), is at least $1 - nt(n) \cdot \frac{1}{nq'(n)t(n)} = 1 - \frac{1}{q'(n)}$.

6.2.2 Proving the `Sender`'s privacy property - The `receiver` gains no computational knowledge about σ_{1-i}

The current setting is analogous to Yao's XOR-lemma. Recall that Yao's XOR-lemma states that given a basic predicate that is "not too easy to predict" (i.e., each polynomial time algorithm may predict the predicate value with probability bounded away from 1), the predicate defined as the XOR of "many" invocations of the basic predicate is unpredictable (i.e., no polynomial time algorithm can predict the predicate value with non-negligible advantage). For details see [GNW95].

On the face of it, the two settings seem somewhat different: here we deal with protocol views, whereas Yao's XOR-lemma deals with predicates. Still, bearing in mind that we are referring to the semi-honest model, the `receiver`'s view of the protocol is simply a random variable, and the other secret (i.e., σ_{1-i}) is a predicate of this view [8]. Our basic predicate receives, as an input, the `receiver`'s view of the subprotocol and returns the value of σ_{1-i} that determined by the view. Notice that the value of σ_{1-i} in the high-level protocol, is (by the definition of the protocol) the XOR of these values (i.e., the values of the σ_{1-i}'s) determined by `receiver`'s views of the different executions of the subprotocol. Therefore, predicting the value of σ_{1-i} from the `receiver`'s view of the high-level protocol, is actually predicting the value of the predicate defined as the XOR of our basic predicate. By the `Sender`'s privacy property of the subprotocol, it is hard to predict well the value of our basic predicate and by Yao's XOR-lemma the claim follows.

More formally, for any fixed i in $\{0,1\}$, let Z_n be a random variable representing the `receiver`'s view of the following *variant of the subprotocol*. In this variant, the `receiver` uniformly selects σ_0 and σ_1 in $\{0,1\}$ (rather than receiving them as input) and the rest of the execution proceeds as usual. Let g be the predicate that assigns to any possible value z of Z_n the value of σ_{1-i} determined by z. We notice that by the `Sender`'s privacy property of the subprotocol (recall that the subprotocol is $\left(\frac{1}{nq'(n)t(n)}, 1-\frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)-WOT$), there is no polynomial time algorithm that predicts $g(Z_n)$ with ad-

[8] Our discussion presuppose that the `receiver`'s view uniquely determines the value of σ_{1-i}. This is not necessarily the case. We may assume, however, that this is the case. For example, we may modify the subprotocol by augmenting a new phase where the `sender` commits itself to the values of σ_0 and σ_1. Clearly, given that the subprotocol is a $\left(\frac{1}{nq'(n)t(n)}, 1-\frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)-WOT$, so is the modified protocol. Moreover, the `receiver`'s view of the modified protocol does uniquely determine the value of σ_{1-i}.

vantage better than $\frac{1}{2} - \frac{1}{2t(n)}$. Let G be the predicate that assigns to any sequence $(z_1, z_2, \ldots, z_{nt(n)})$, where each of the z's is taken from Z_n, the value $\bigoplus_{j=1}^{nt(n)} g(z_j)$. By the weak unpredictability of g and (the uniform version of) Yao's XOR-lemma, there is no polynomial time algorithm that predicts $G(z_1, z_2, \ldots, z_{nt(n)})$ with advantage greater than $\frac{1}{2} \cdot (1 - \frac{1}{t(n)})^{nt(n)} + neg(n) \approx neg(n)$. (Note that in order to apply the uniform version of Yao's XOR-lemma, we need to have the ability to sample Z_n, but this can be done by simulating both sides of the subprotocol). Now, the point is that by the definition of G and the distribution of its inputs, predicting G is identical to predicting the value of σ_{1-i} from the `receiver`'s view the following *variant of the high-level protocol.* In this variant, the `receiver` uniformly selects σ_0 and σ_1 in $\{0, 1\}$ (rather than receiving them as input), and the rest of the execution proceeds as usual. Moreover, it is easy to see that, predicting the value of σ_{1-i} given `receiver`'s view of the above variant, is as hard as in the (original version of the) high-level protocol. Therefore, predicting σ_{1-i} in the high-level protocol is as hard as predicting G, and the claim follows.

6.2.3 Proving the `Receiver`'s privacy property - The `sender` does not gain more information about i than $\frac{1}{q'(n)}$

The `sender`'s view of the high-level protocol is a concatenation of $nt(n)$ views of the subprotocol (i.e., $\left(\frac{1}{nq'(n)t(n)}, 1 - \frac{1}{t(n)}, \frac{1}{nq'(n)t(n)}\right)$–$WOT$). Therefore, the statistical difference between the `sender`'s views of the high-level protocol in case $i = 0$ and the case $i = 1$, is at most $nt(n)$ times the statistical difference between the `sender`'s views of the subprotocol in these cases. Recalling that the latter is $\frac{1}{nq'(n)t(n)}$, we are done.

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

Recall that q'' is any positive polynomial.

In the following protocol the `sender` repeatedly transfers σ_i to the `receiver`, using $\left(\frac{1}{nq''(n)}, neg(n), \frac{1}{nq''(n)}\right)$–$WOT$ and the `receiver` determine the correct value using majority rule. The point is to decrease the probability that the `receiver` wrongly determines σ_i.

7.1 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. The `sender` transfers σ_i for n times to the `receiver`, using $\left(\frac{1}{nq''(n)}, neg(n), \frac{1}{nq''(n)}\right)-WOT$.

   That is, the parties invoke $\left(\frac{1}{nq''(n)}, neg(n), \frac{1}{nq''(n)}\right)-WOT$ for n times, such that in each invoking the `sender`'s secrets are σ_0 and σ_1 and the `receiver`'s index is i.

2. The `receiver` decides the value of σ_i by majority rule.

   Motivation: The probability that the `receiver` obtains the right value of σ_i in a single iteration is very (i.e., at least $1 - \frac{1}{nq''(n)}$). Therefore, the probability that through n independent iteration, the `receiver` obtains the right value of σ_i in at least half of them, is overwhelming.

7.2 Analysis

In the following analysis we refer to the above protocol as the *high-level protocol*, and we refer to $\left(\frac{1}{nq''(n)}, neg(n), \frac{1}{nq''(n)}\right)-WOT$ as the *subprotocol.*

7.2.1 Proving the Correctness property - The probability that the `receiver` does not outputs σ_i is negligible

The proof is immediate by Chernoff bound and the Correctness property of the subprotocol.

7.2.2 Proving the `Sender`'s privacy property - The `receiver` gains no computational knowledge about σ_{1-i}

Intuitively, since the `receiver`'s view of the high-level protocol is a concatenations of n `receiver`'s views of the subprotocol, the computational knowledge one might receive from the `receiver`'s view of the high-level protocol is not more than n times the computational knowledge one might receive from the `receiver`'s view of the subprotocol. Hence, the `receiver` gains no more computational knowledge about σ_{1-i} than n times negligible, and therefore no more than negligible.

The actual proof is by a hybrid argument. We assume, by contradiction, the existence of a polynomial time algorithm, A, that has non-negligible advantage in predicting σ_{1-i} from the `receiver`'s view of the high-level protocol. For any $k \in \{0, 1, \ldots, n\}$, the k-hybrid is defined as the `receiver`'s view of the following variant of the high-level protocol. The variant consists of k executions of the subprotocol with σ_{1-i} set to 0, follows by $n - k$ executions of the subprotocol with σ_{1-i} set to 1. Note that the extreme hybrids are the `receiver`'s views of the high-level protocol with the different values of σ_{1-i}. (i.e., the 0-hybrid is the `receiver`'s view of the high-level protocol where $\sigma_{1-i} = 0$, and the n-hybrid is the `receiver`'s view where $\sigma_{1-i} = 1$). By the contradiction assumption, A distinguishes between the two extremes hybrids with non-negligible success, and therefore, A distinguishes between two neighboring hybrids with non-negligible success. Note that any two neighboring hybrids, differ only in the value of σ_{1-i} in one of their subprotocol executions. Hence, this protocol can predict with non-negligible advantage, the value of σ_{1-i} from the `receiver`'s view of the subprotocol, a contradiction to the `Sender`'s privacy property of the subprotocol.

7.2.3 Proving the `Receiver`'s privacy property - The `sender` does not gain more information about i than $\frac{1}{q''(n)}$

This analysis is analogous to the one given in Subsection 6.2.3.

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

The following protocol follows the protocol presented by Crépeau and Kilian [CK90], and is presented here for the sake of self-containment (since the original paper does not contain the analysis of the protocol). The protocol uses a kind of secret sharing applied to the `receiver` request (i.e., i) in order to reduce the probability that the `sender` learns the request through the protocol's execution.

8.1 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. The `receiver` selects $\mu_1, \mu_2, \ldots, \mu_{n-1}$ uniformly in $\{0,1\}$, and sets μ_n to $i \oplus (\bigoplus_{j=1}^{n-1} \mu_j)$.

   Motivation: The `receiver` "splits" the value of i among the different μ's such that $\bigoplus_{j=1}^{n} \mu_j = i$. Thus, in order to know i, one should know the values of all $\mu_1, \mu_2, \ldots, \mu_n$.

2. The `sender` selects $\omega_{0,1}, \omega_{0,2}, \ldots, \omega_{0,n-1}$ uniformly in $\{0,1\}$ and sets $\omega_{0,n}$ to $\sigma_0 \oplus (\bigoplus_{j=1}^{n-1} \omega_{0,j})$.

3. For all $1 \leq j \leq n$, the `sender` sets $\omega_{1,j}$ to $(\omega_{0,j} \oplus \sigma_0 \oplus \sigma_1)$.

   Motivation: The `sender` "splits" its two secrets among the different ω's such that $\bigoplus_{j=1}^{n} \omega_{0,j} = \sigma_0$ and for each j, $\omega_{1,j} = \omega_{0,j} \oplus \sigma_0 \oplus \sigma_1$. Thus, in order to know σ_k (where k is in $\{0,1\}$) one should know any sequence of secrets $\omega_{x_1,1}, \omega_{x_2,2} \ldots, \omega_{x_n,n}$ with the property that $\bigoplus_{j=1}^{n} x_j = k$.

4. For all $1 \leq j \leq n$, the `sender` transfers $\omega_{\mu_j,j}$ to the `receiver` using $\big(neg(n), neg(n), \frac{1}{3}\big)-WOT$.

   That is, the parties invoke $\big(neg(n), neg(n), \frac{1}{3}\big)-WOT$ for n times such that in the j'th invoking the `sender`'s secrets are $\omega_{0,j}$ and $\omega_{1,j}$ and the `receiver`'s index is μ_j.

5. The `receiver` locally outputs $\bigoplus_{j=1}^{n} \omega_{\mu_j,j}$.

8.2 Analysis

In the following analysis we refer to the above protocol as the *high-level protocol*, and we refer to $(neg(n), neg(n), \frac{1}{3})-WOT$ as the *subprotocol*.

8.2.1 Proving the Correctness property - The probability that the `receiver` does not outputs σ_i is negligible

By the Correctness property of $\big(neg(n), neg(n), \frac{1}{3}\big) - WOT$ we have that, except for a negligible probability, the `receiver` receives $\omega_{\mu_1,1}, \omega_{\mu_2,2} \ldots, \omega_{\mu_n,n}$

correctly. Therefore the `receiver` outputs

$$\begin{aligned}
\bigoplus_{j=1}^{n} \omega_{\mu_j,j} &= \left(\bigoplus_{j=1}^{n} \omega_{0,j}\right) \oplus \left(\bigoplus_{1\le j\le n, \mu_j=1} (\sigma_0 \oplus \sigma_1)\right) \\
&= \sigma_0 \oplus \begin{cases} (\sigma_0 \oplus \sigma_1) & \text{if } \bigoplus_{j=1}^{n} \mu_j = 1, \\ 0 & \text{otherwise.} \end{cases} \\
&= \begin{cases} \sigma_1 & \text{if } \bigoplus_{j=1}^{n} \mu_j = 1, \\ \sigma_0 & \text{otherwise.} \end{cases} \\
&= \sigma_i
\end{aligned}$$

8.2.2 Proving the `Sender`'s privacy property - The `receiver` does not gain computational knowledge about σ_{1-i}

We prove an equivalent claim by which the `receiver` does not gain computational knowledge about the value of $\sigma_0 \oplus \sigma_1$. In particular we assume that it is impossible the predict the value of $\sigma_0 \oplus \sigma_1$ from the `receiver`'s view of the subprotocol, and we prove that it is impossible the predict the value of $\sigma_0 \oplus \sigma_1$ from the `receiver`'s view of the high-level protocol.

For simplicity we do not prove the claim directly for the *high-level protocol*, but rather for the randomized variant of the *high-level protocol*, where the `receiver` uniformly selects σ_0 and σ_1 in $\{0, 1\}$ (rather than receiving them as input) and the rest of the execution proceeds as usual [9]. The proof uses a hybrid argument. We assume by contradiction the existence of a polynomial time algorithm, A, that for some fixed $i \in \{0, 1\}$, has non-negligible advantage in predicting $\sigma_0 \oplus \sigma_1$ from the `receiver`'s view of the above variant. For $k \in \{0, 1, \ldots, n\}$, the k-hybrid is defined as the `receiver`'s view of the following protocol: This protocol is the same as the randomized variant of the high-level protocol described above, except for Step 3. In this protocol version of Step 3, the last $n - k$ $\omega_{1,\cdot}$'s are uniformly chosen (rather than set to $\omega_{1,\cdot} \oplus \sigma_0 \oplus \sigma_1$). Formally, the modified version of Step 3 is:

3.′ For all $1 \le j \le k$, the `sender` sets $\omega_{1,j}$ to $(\omega_{0,j} \oplus \sigma_0 \oplus \sigma_1)$,
for all $k + 1 \le j \le n$, the `sender` selects $\omega_{1,j}$ uniformly in $\{0, 1\}$.

[9] It is easy to see that the `Sender`'s privacy property of this variant is the same as of the original protocol.

We note that one extreme hybrid (the n-hybrid) is the above variant of the high-level protocol and therefore, by the contradiction assumption, A has non-negligible advantage in predicting $\sigma_0 \oplus \sigma_1$ from the `receiver`'s view of this hybrid. We also note that the other extreme hybrid (the 0-hybrid) is a protocol that is independent of the value of σ_1 and therefore A has no advantage in predicting the value of $\sigma_0 \oplus \sigma_1$ from the `receiver`'s view of this hybrid. Hence, there exists two neighboring hybrids with a non-negligible gap between the advantage A has in predicting the value of $\sigma_0 \oplus \sigma_1$ from the `receiver`'s views of the two hybrids. Thus, A distinguishes, with non-negligible success, between the `receiver`'s views of two protocols, which differ only in the inputs given to a single execution of the subprotocol. In one of the protocols the value of $\sigma_0 \oplus \sigma_1$ (in the call to the subprotocol) is equal to this value in the protocol itself, where in the other protocol this value is randomly chosen. Thus A can be used to contradict the `Sender`'s privacy property of the subprotocol.

8.2.3 Proving the `Receiver`'s privacy property - The `sender` does not gain non-negligible information about i

For any σ'_0, σ'_1 and $i' \in \{0,1\}$, let $D_{i'}^{\sigma'_0,\sigma'_1}$ be the random variable that represents the `sender`'s view of the subprotocol with inputs: σ'_0, σ'_1 and i' (σ'_0 and σ'_1 are the `sender`'s secrets, and i' is the `receiver`'s index). Note that by the `Receiver`'s privacy property of the subprotocol (recall that the subprotocol is $\left(neg(n), neg(n), \frac{1}{3}\right) - WOT$), the statistical difference between $D_0^{\sigma'_0,\sigma'_1}$ and $D_1^{\sigma'_0,\sigma'_1}$ is smaller than $\frac{1}{3}$.

In the following discussion, we fixed the bits $\upsilon_0^1, \upsilon_0^2, \ldots, \upsilon_0^n, \upsilon_1^1, \upsilon_1^2, \ldots, \upsilon_1^n$ to some values in $\{0,1\}$. For any $i \in \{0,1\}$, let Z_i be the random variable that represents the `sender`'s view of the following variant of the high-level protocol. In this variant, the `receiver`'s index is set to i, and the `sender` shares are fixed to $\upsilon_0^1, \upsilon_0^2, \ldots, \upsilon_0^n, \upsilon_1^1, \upsilon_1^2, \ldots, \upsilon_1^n$ (i.e., ω_i^j is set to υ_i^j). We prove that the statistical difference between Z_0 and Z_1 is negligible, and since $\upsilon_0^1, \upsilon_0^2, \ldots, \upsilon_0^n, \upsilon_1^1, \upsilon_1^2, \ldots, \upsilon_1^n$ were chosen arbitrarily, the proof of the `Receiver`'s privacy property follows.

The proof stems from the following claim, which is an immediate extension of [SV97, Prop. 3.6]:

Claim 8.2.1 *Let $\{X_0^1, X_0^2, \ldots, X_0^n\}$ and $\{X_1^1, X_1^2, \ldots, X_1^n\}$ be two sequences of independent random variables, and let Y_k, for both $k \in \{0,1\}$, be the fol-*

lowing random variable:
Y_k: Choose $m_1, m_2, \ldots, m_n$ uniformly in $\{0,1\}$ such that $\left(\bigoplus_{j=1}^{n} m_j\right) = k$. Output a sample of $(X^1_{m_1}, X^2_{m_2}, \ldots, X^n_{m_n})$.

Then

$$stat(Y_0, Y_1) = \prod_{j=1}^{n} stat(X^j_0, X^j_1)$$

Applying Claim 8.2.1 to the sequence $\left\{D_0^{v_0^1,v_1^1}, D_0^{v_0^2,v_1^2}, \ldots, D_0^{v_0^n,v_1^n}\right\}$ and $\left\{D_1^{v_0^1,v_1^1}, D_1^{v_0^2,v_1^2}, \ldots, D_1^{v_0^n,v_1^n}\right\}$, and the corresponding Z_0 and Z_1, we get that

$$stat(Z_0, Z_1) = \prod_{j=1}^{n} stat(D_0^{v_0^j,v_1^j}, D_1^{v_0^j,v_1^j})$$

By the fact that for all $j \in \{1, 2, \ldots, n\}$, $stat(D_0^{v_0^j,v_1^j}, D_1^{v_0^j,v_1^j}) < \frac{1}{3}$, we get that $stat(Z_0, Z_1) < (\frac{1}{3})^n$, and we are done.

9 Further issues

A natural question to ask is whether a similar result can be obtained even if the permutation requirement is somewhat relaxed. For example can we construct an OT based on dense collection of one-to-one one-way functions (i.e., the domains of the functions contain polynomial fractions of all strings of one length and the ranges of the functions contain polynomial fractions of all strings of another length)? 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, but the size of the function range is 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), then the above result still holds. The reason being that such a collection can be transformed into a dense length-preserving collection, by padding, without lost of generality, the domain elements with $m - n$ zeros.

in the first step of our implementation (i.e., Subsection 5.3 Step 2) s is uniformly chosen in $\{0,1\}^n$ and r_{1-i} is uniformly chosen in $\{0,1\}^m$ (rather than selecting both elements uniformly in $\{0,1\}^n$) and the rest of the implementation proceeds as usual.

10 Acknowledgement

I would like to thank my advisor, Oded Goldreich, for suggesting the question investigated in this work, and for contributing many of the essential ideas used in this thesis. He also, through numerous meetings, helped me to turn my rough ideas into formal proofs.

References

[BCR86] G. Brassard, C. Crepeau, and J.-M. Robert. Information theoretic reductions among disclosure problems. In *27th Annual Symp. on Foundations of Computer Science (FOCS '86)*, pages 168–173, Los Angeles, Ca., USA, October 1986. IEEE.

[Blu83] Manuel Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1(2):175–193, May 1983.

[CK90] C. Crépeau and J. Kilian. Weakening security assumptions and oblivious transfer. In *Advances in Cryptology (CRYPTO '88)*, pages 2–7, Berlin - Heidelberg - New York, August 1990. Springer.

[Cré87] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In Carl Pomerance, editor, *Advances in Cryptology—CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 350–354. Springer-Verlag, 1988, 16–20 August 1987.

[CS91] C. Crépeau and M. Sántha. On the reversibility of oblivious transfer. In Donald W. Davies, editor, *Proceedings of Advances in Cryptology (EUROCRYPT '91)*, volume 547 of *LNCS*, pages 106–113, Berlin, Germany, April 1991. Springer.

[DKS99] 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*, 1592:56–??, 1999.

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[GKM+00] Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan. The relationship between public key encryption and oblivious transfer. In IEEE, editor, *41st Annual Symp. on Foundations of Computer Science: proceedings: 12–14 November, 2000, Redondo Beach, California*, pages 325–335. IEEE, 2000.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In ACM, editor, *Proceedings of the twenty-first annual ACM Symp. on Theory of Computing, Seattle, Washington, May 15–17, 1989*, pages 25–32, New York, NY, USA, 1989. ACM Press.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 218–229, 1987.

[GNW95] Goldreich, Nisan, and Wigderson. On yao's XOR-lemma. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1995.

[Gol99] Oded Goldreich. *Modern cryptography, probabilistic proofs, and pseudorandomness*, volume 17 of *Algorithms and combinatorics*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1999.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, UK, 2001.

[Gol02] Oded Goldreich. Foundations of cryptography - volume 2. Working Draft, available at www.wisdom.weizmann.ac.il/oded/foc-vol2.html, 2002.

[Rab81] M. O. Rabin. How to exchange secrets by oblivious transfer. TR-81, Harvard, 1981.

[SV97] Amit Sahai and Salil Vadhan. A complete promise problem for statistical zero-knowledge. In *Proceedings of the 38th Annual Symp. on the Foundations of Computer Science*, pages 448–457. IEEE, October 1997.

[Yao86] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Symp. on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE Computer Society Press, 1986.