Extending Oblivious Transfers Efficiently

Yuval Ishai[1], Joe Kilian[2], Kobbi Nissim[2*], and Erez Petrank[1**]

[1] Department of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel. {yuvali|erez}@cs.technion.ac.il

[2] NEC Laboratories America, 4 Independence Way, Princeton, NJ 08550, USA. {joe|kobbi}@nec-labs.com

Abstract. We consider the problem of extending oblivious transfers: Given a small number of oblivious transfers "for free," can one implement a large number of oblivious transfers? Beaver has shown how to extend oblivious transfers given a one-way function. However, this protocol is inefficient in practice, in part due to its non-black-box use of the underlying one-way function.

We give efficient protocols for extending oblivious transfers in the random oracle model. We also put forward a new cryptographic primitive which can be used to instantiate the random oracle in our constructions. Our methods suggest particularly fast heuristics for oblivious transfer that may be useful in a wide range of applications.

1 Introduction

Is it possible to base oblivious transfer on one-way functions? Partial answers to this question were given by Impagliazzo and Rudich [22] and Beaver [1]. Impagliazzo and Rudich [22] showed that a *black-box* reduction from oblivious transfer to a one-way function (or a one-way permutation) would imply P$\neq$NP. They gave an oracle that combines a random function and a PSPACE oracle and proved that relative to this oracle one-way functions exist, but secret-key agreement is impossible. In other words, even an *idealized* one-way function (a random oracle) is insufficient for constructing secret-key agreement and hence oblivious transfer. A number of papers have continued this line of research and drew the limits of black-box reductions in cryptography, mapping the separations between the power of cryptographic primitives in relativized worlds [34, 15, 16, 25, 14].

It is not known whether a *non-black-box* reduction from oblivious transfer to one-way functions exists. Impagliazzo and Rudich's result strongly suggests that with the current knowledge in complexity theory we cannot base oblivious transfer on one-way functions. However, a remarkable theorem of Beaver [1] shows that a 'second-best' alternative *is* achievable – one-way functions are sufficient to *extend* a few oblivious transfers into many, i.e. it is possible to implement a large number of oblivious transfers given just a small number of oblivious transfers:

* Work partially done while the second author was at DIMACS, Rutgers University, 96 Frelinghuysen Road Piscataway, NJ 08854, USA.

** This research was supported by the E. AND J. BISHOP RESEARCH FUND.

Theorem 1 ([1]). *Let k be a computational security parameter. If one-way functions exist, then for any constant $c > 1$ there exists a protocol for reducing k^c oblivious transfers to k oblivious transfers.*

Interestingly, Beaver's reduction is inherently non-black-box with respect to the one-way function it uses.

The results of Impagliazzo and Rudich and Beaver are motivated by both theory and practice. From a theoretical point of view, one is interested in the weakest assumptions needed for oblivious transfer, and the type of reductions employed. From a practical point of view, oblivious transfer protocols are based on public-key primitives, which seem to require more structure than private-key primitives, and in practice are more expensive to implement. Alternative physical or multi-party implementations of oblivious transfer are also comparatively expensive. Thus, it is highly desirable to obtain methods for implementing oblivious transfers with (amortized) cost comparable to that of *private-key* primitives.

Beaver's protocol shows how to implement most of one's oblivious transfers using simple primitives. Thus, if public-key primitives simply did not exist, one could implement a few oblivious transfers using, for example, multi-party computation, and then use one-way functions for the rest.

Unfortunately, Beaver's protocol appears to be inefficient in practice. In particular it requires that operations be performed for every gate of a circuit computing, among other things, a pseudo-random generator. Consequently, the protocol requires work at least quadratic in the circuit complexity of the pseudo-random generator.

1.1 Oblivious Transfer

Oblivious transfer (OT) [32, 10, 6, 23] is a ubiquitous cryptographic primitive that may be used to implement a wide variety of other cryptographic protocols, including secret key exchange, contract signing [10], and secure function evaluation [36, 19, 20, 23].

Oblivious transfer is a two-party protocol between a *sender* and a *receiver*. Several flavors of OT have been considered and shown equivalent [9, 6]. In the most useful type of OT, often denoted $\binom{2}{1}$-OT [10] (for the single-bit version) or ANDOS [6] (for the multi-bit version), the sender holds a pair of strings[3] and the receiver holds a selection bit. At the end of the protocol the receiver should learn just the selected string, and the sender should not gain any new information. Moreover, it is required by default that the above properties hold even if the sender or the receiver maliciously deviate from the protocol. This notion of OT can be conveniently formalized within the more general framework

[3] In a typical application of OT these strings are used as key material, in which case their length should be equal to a cryptographic security parameter. Most direct implementations of OT (cf. [28]) in fact realize OT with these parameters. Moreover, OT of arbitrarily long strings can be reduced to such OT by making a simple use of a pseudo-random generator (see Appendix B). Hence, the following discussion is quite insensitive to the length of the strings being transferred.

of secure computation (see Section 2). In fact, OT is a *complete* primitive for general secure computation [20, 23].

Efficiency is particularly crucial for oblivious transfer, due to its massive usage in secure protocols. For instance, general protocols for secure computation (e.g., [36, 19, 23]) require at least one OT invocation per input bit of the function being evaluated. This is also the case for more specialized or practically-oriented protocols (e.g., [30, 29, 17, 26, 12, 27]), where oblivious transfers typically form the efficiency bottleneck.

1.2 Our Results

In light of the state of affairs described above, it is quite possible that one cannot extend oblivious transfers in a black-box manner. It was also possible, in analogy to [22, 1], that even a random oracle cannot be used to extend oblivious transfers.We show that this is *not* the case. Specifically, we show the following black-box analogue of Theorem 1:

Theorem 2 (Main Theorem). *Let k be a computational security parameter. For any constant $c > 1$, there exists a protocol* in the random oracle model *for reducing k^c oblivious transfers to k oblivious transfers. Alternatively, the random oracle can be replaced by a black-box use of a* correlation robust *hash function, as defined in Definition 1, Section 5.*

We note that our result for the random oracle model is actually stronger. For any $\epsilon > 0$, we can reduce $2^{k^{1-\epsilon}}$ oblivious transfers to k oblivious transfers of k-bit strings. This reduction is essentially tight, in the sense that the negative result of [22] can be extended to rule out a similar reduction of $2^{\Omega(k)}$ oblivious transfers to k oblivious transfers (assuming that the adversary is allowed polynomial time in the number of oblivious transfers).

Contribution perspective. It is instructive to draw an analogy between the problem of extending oblivious transfers and that of extending *public-key encryption.* To encrypt a *long* message (or multiple messages) using a public-key encryption scheme, it suffices to encrypt a *short* secret key κ for some private-key scheme, and then encrypt the long message using the private-key scheme with key κ. Thus, public-key encryption can be readily extended via a black-box use of a (far less costly) private-key primitive. The existence of such an efficient black-box reduction has a significant impact on our everyday use of encryption. The aim of the current work is to establish a similar result within the richer domain of secure computations.

Efficiency. Our basic protocol is extremely efficient, and requires each party to make a small constant number of calls to the random oracle for each OT being performed. All other costs (on top of the initial seed of k OTs) are negligible. This basic protocol, however, is insecure against a malicious receiver who may deviate from the protocol's instructions. To obtain a fully secure protocol, we employ a cut-and-choose technique. This modification increases the cost of the basic protocol by a factor of σ, where σ is a statistical security parameter.

Specifically, any "cheating" attempt of a malicious receiver will be detected by the sender, except with probability $2^{-\Omega(\sigma)}$. Thus, in scenarios where a penalty is associated with being caught cheating, this (rather modest) cut-and-choose overhead can be almost entirely eliminated. We also discuss some optimized variants of our protocols, in particular ones that are tailored to an "on-line" setting where the number of desired oblivious transfers is not known in advance.

Related work. There is a vast literature on implementing OT in various settings and under various assumptions. Since OT implies key exchange, all these OT protocols require the use of expensive public-key operations. The problem of *amortizing* the cost of multiple oblivious transfers has been considered by Naor and Pinkas [28]. Their result is in a sense complementary to ours: while the savings achieved are modest, the amortization "kicks in" quite early. Thus, their techniques can be used for reducing the cost of the seed of k oblivious transfers required by our protocols.

1.3 On the Use of a Random Oracle

We describe and analyze our protocols in the random oracle model. Such an analysis is used in cryptography for suggesting the feasibility of protocols, and for partially analyzing protocols (in particular, practical ones). Instead of making purely heuristic arguments, one considers an idealized hash function, and proves rigorous results in this "nearby" version of the problem. The latter approach is advocated e.g. by Bellare and Rogaway [3]. They suggest to analyze protocols with hash functions modeled by an idealized version, implemented by some magic black box. For example, instead of trying to analyze a protocol that uses a specific hash function $h(x) : \{0,1\}^{2k} \rightarrow \{0,1\}^k$, one analyzes the system that uses a random function $H : \{0,1\}^{2k} \rightarrow \{0,1\}^k$, chosen uniformly from the space of all such functions.

All parties being considered, both legitimate parties and adversaries, are given access to H as a black box (or oracle). The protocol can instruct the parties to make queries to H. The adversary attacking the protocol is allowed to make arbitrary queries to H, at unit cost, but is bounded in how many queries it is allowed to make (if only by a bound on its running time). Using the complete lack of structure of H and the fact that the adversary can explore only a tiny portion of its inputs allows us to rigorously analyze idealized versions of systems that in vanilla form resist all analysis. The heuristic leap of faith is that the real world matches the idealized world insofar as security is concerned, i.e., that the hash function h is sufficiently "structureless" that the system remains secure if it uses h in place of H.

This approach has been applied to many practical systems. To mention just a few, it was used for optimal asymmetric encryption (OAEP) [5, 33], for replacing interaction in the Fiat-Shamir signature scheme [13], and to justify hashing methodologies [4, 35], a method for basing cryptosystems on one-way permutations [11], for analyzing the DESX construction [24], as well as RSA-based signature schemes with hashing [3]. In practical implementations, highly efficient hash functions such as SHA1 or RC5 are typically used to instantiate the random oracle.

There is no rigorous methodology allowing one to safely replace the hash function with a cryptographic instantiation. Indeed, Canetti, Goldreich and Halevi [8] give an explicit example of a cryptographic protocol that is secure using an ideal random function H, but is insecure for any polynomial-time computable instantiation of H. Their counterexample stems from the difference between an efficiently computable function and a random oracle: The efficient function has a small circuit whereas the random oracle does not. Further research along these lines appears in [31, 21, 2].

We would like to stress, however, that these concerns can be at least partially dismissed in the context of the current work. First, in our security proofs we treat the random oracle as being *non-programmable* [31], i.e., we do not allow the simulators used in these proofs to fake the answers received from the oracle. This version of the random oracle model is more conservative, in that it rules out some of the above counterexamples. More importantly, we provide an *alternative* to the random oracle model by suggesting a concrete and seemingly natural cryptographic primitive that can be used to instantiate our protocols. Specifically, we require an explicit $h : \{0,1\}^* \to \{0,1\}$ so that for a random and independent choice of (polynomially many) strings $s, t_1, \ldots, t_m \in \{0,1\}^k$, the joint distribution $(h(s \oplus t_1), \ldots, h(s \oplus t_m), t_1, \ldots, t_m)$ is pseudo-random. Since this is a *simple* property enjoyed by a random function, any evidence that hash functions such as SHA1 or RC5 violate this property could be considered a valid attack against these functions.

1.4 Road Map

Section 2 contains some preliminaries and basic tools. We give the basic version of our protocol in Section 3. This protocol is insecure against a malicious receiver. A modified version of the basic protocol that achieves full security is described in Section 4. In Section 5 we describe a primitive that captures a property of the random oracle that is sufficient for our constructions. We conclude with some open problems.

2 Preliminaries

2.1 Secure Two-Party Computation

It is convenient to define our problem within the more general framework of secure two-party computation. In the following we assume the reader's familiarity with standard simulation-based definitions of secure computation from the literature. For self-containment, the necessary definitions are sketched in Appendix A.

The use of a random oracle. When augmenting the standard definition of two-party computation with a random oracle H, we assume that the oracle is picked at random in both the real function evaluation process involving the adversary and the ideal process involving the simulator. Furthermore, we require that the simulation be indistinguishable from the real execution even if the distinguisher is allowed to adaptively make polynomially many queries to the *same* H that was used in the process (real or ideal) generating its input. This is especially significant when simulating a semi-honest (i.e., passively corrupted) party, since this party by definition cannot make any additional calls to H. However, even when simulating a malicious (i.e., actively corrupted) party, who can make the additional calls himself, this requirement is important for ensuring that the transcripts provided by the simulator are indeed consistent with the instance of H to which it had access. This is referred to as the *non-programmable* random oracle model.

Reductions. It is often convenient to design secure protocols in a modular way: first design a high level protocol assuming an idealized implementation of a lower level primitive, and then substitute a secure implementation of this primitive. If the high level protocol securely realizes a functionality f and the lower level primitive a functionality g, the high level protocol may be viewed as a secure *reduction* from f to g. Such reductions can be formalized using a *hybrid* model, where the parties in the high level protocol are allowed to invoke g, i.e., a trusted party to which they can securely send inputs and receive the corresponding outputs. In our case, we will be interested in reductions where f implements "many" oblivious transfers and g "few" oblivious transfers. Moreover, we will usually restrict our attention to reductions making use of a *single* invocation to g. Appropriate composition theorems, e.g. from [7], guarantee that the call to g can be replaced by any secure[4] protocol realizing g, without violating the security of the high level protocol. Moreover, these theorems relativize to the random oracle model. Thus, it suffices to formulate and prove our reductions using the above hybrid model.

Black-box reductions. The above framework for reductions *automatically* guarantees that the high-level protocol for f make a black-box use of g, in the sense that both the protocol and its security proof do not depend on the implementation of g. Thus, all our reductions in the random oracle model are fully black box. For comparison, Beaver's reduction [1] is clearly non-black-box with respect to the one-way function (or PRG) on which it relies. We refer the reader to [15, 16] for a more thorough and general exposition to black-box reductions in cryptography.

2.2 Oblivious Transfer

An OT protocol can be defined as a secure two-party protocol between a sender S and a receiver R realizing the OT functionality (see Appendix A) . This definition implies the properties mentioned in the introduction: the sender cannot learn the receiver's selection, and the receiver cannot learn more than one of the two strings held by the sender. It is interesting to note that this definition imposes additional desirable features. For instance, a malicious sender must "know" a

[4] Here one should use an appropriate notion of security, e.g., the one from [7].

pair of string to which the receiver's selection effectively applies.[5] This may be important for using OT in the context of other protocols.

The most general OT primitive we consider, denoted OT_ℓ^m, realizes m (independent) oblivious transfers of ℓ-bit strings. That is, OT_ℓ^m represents the following functionality:

Inputs: S holds m pairs $(x_{j,0}, x_{j,1})$, $1 \le j \le m$, where each $x_{j,b}$ is an ℓ-bit string. R holds m selection bits $\mathbf{r} = (r_1, \ldots, r_m)$.
Outputs: R outputs x_{j,r_j} for $1 \le j \le m$. S has no output.

The task of extending oblivious transfers can be defined as that of reducing OT_ℓ^m to OT_k^k, where k is a security parameter and $m > k$. (For simplicity, one may think of ℓ as being equal to k.) That is, we would like to implement m oblivious transfers of ℓ-bit strings by using k oblivious transfers of k-bit strings. However, it will be more convenient to reduce OT_ℓ^m to OT_m^k. The latter primitive, in turn, can be very easily reduced to OT_k^k by generating $2m$ pseudo-random bits.[6] This reduction proceeds by performing OT of k pairs of independent seeds, and then sending the k pairs of strings masked with the outputs of the generator on the corresponding seeds. See Appendix B for a formal description.

We note that using known reductions, OT_k^k can be further reduced to k invocations of OT_k^1 (the type of OT typically realized by direct implementations) or $O(k^2)$ invocations of OT_1^1 [6]. Thus, any secure implementation for the most basic OT variant can be used as a black box to securely realize our high level OT_ℓ^m protocol. We prefer to use OT_k^k as our cryptographic primitive for extending oblivious transfers, due to the possibility for a more efficient direct implementation of this primitive than via k separate applications of OT_k^1 [28].

2.3 Notation

We use capital letters to denote matrices and small bold letters to denote vectors. We denote the jth row of a matrix M by $\mathbf{m}_j$ and its ith column by $\mathbf{m}^i$. The notation $b \cdot \mathbf{v}$, where b is a bit and $\mathbf{v}$ is a binary vector, should be interpreted in the natural way: it evaluates to 0 if $b = 0$ and to $\mathbf{v}$ if $b = 1$.

3 Extending OT with a Semi-Honest Receiver

In this section we describe our basic protocol, reducing OT_ℓ^m to OT_m^k. As noted above, this implies a reduction to OT_k^k with a very small additional cost. The security of this protocol holds as long as the receiver is semi-honest. We will later modify the protocol to handle malicious receivers. The protocol is described in Fig. 1.

5 This follows from the fact that a malicious sender's output should be simulated *jointly* with the honest receiver's output.

6 A pseudo-random generator can be easily implemented in the random oracle model without additional assumptions.

INPUT OF S: m pairs $(x_{j,0}, x_{j,1})$ of ℓ-bit strings, $1 \leq j \leq m$.
INPUT OF R: m selection bits $\mathbf{r} = (r_1, \ldots, r_m)$.
COMMON INPUT: a security parameter k.
ORACLE: a random oracle $H : [m] \times \{0,1\}^k \to \{0,1\}^\ell$.
CRYPTOGRAPHIC PRIMITIVE: An ideal OT_m^k primitive.

1. S initializes a random vector $\mathbf{s} \in \{0,1\}^k$ and R a random $m \times k$ bit matrix T.
2. The parties invoke the OT_m^k primitive, where S acts as a receiver with input $\mathbf{s}$ and R as a sender with inputs $(\mathbf{t}^i, \mathbf{r} \oplus \mathbf{t}^i)$, $1 \leq i \leq k$.
3. Let Q denote the $m \times k$ matrix of values received by S. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.) For $1 \leq j \leq m$, S sends $(y_{j,0}, y_{j,1})$ where $y_{j,0} = x_{j,0} \oplus H(j, \mathbf{q}_j)$ and $y_{j,1} = x_{j,1} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$.
4. For $1 \leq j \leq m$, R outputs $z_j = y_{j,r_j} \oplus H(j, \mathbf{t}_j)$.

Fig. 1. Extending oblivious transfers with a semi-honest receiver.

It is easy to verify that the protocol's outputs are correct (i.e., $z_j = x_{j,r_j}$) when both parties follow the protocol.

EFFICIENCY. The protocol makes a single call to OT_m^k. In addition, each party evaluates at most $2m$ times (an implementation of) a random oracle mapping $k + \log m$ bits to ℓ bits. All other costs are negligible. The cost of OT_m^k is no more than k times the cost of OT_k^1 (the type of OT realized by most direct implementations) plus a generation of $2m$ pseudo-random bits. In terms of round complexity, the protocol requires a single message from S to R in addition to the round complexity of the OT_m^k implementation.

3.1 Security

We prove that the protocol is secure against a malicious sender and against a semi-honest receiver. More precisely, we show a *perfect* simulator for any malicious sender S^* and a *statistical* simulator for R. In the latter case, the output of the ideal process involving the simulator is indistinguishable from that of the real process even if the distinguisher is allowed to adaptively make $2^k/k^{\omega(1)}$ additional calls to H.

We let TP denote the trusted party for the OT_ℓ^m functionality in the ideal function evaluation process.

Simulating S^*. It is easy to argue that the output of an arbitrary S^* can be perfectly simulated. Indeed, all S^* views throughout the protocol is a k-tuple of uniformly random and independent vectors, received from the OT_m^k primitive in Step 2. This guarantees that the receiver's selections remain perfectly private. However, as discussed above, the OT definition we are using is stronger in the sense that it requires to simulate the output of the malicious sender jointly with the output of the honest receiver. Such a simulator for a malicious sender S^* may proceed as follows:

- Run S^* with a uniformly chosen random input ρ. Let $\mathbf{s}^*$ be the input S^* sends to the OT_m^k primitive in Step 2. Generate a random $m \times k$ matrix Q, and feed S^* with the columns of Q as the reply from the OT_m^k primitive.
- Let $(y_{j,0}^*, y_{j,1}^*)$ be the messages sent by S^* in Step 3. Call TP with inputs $x_{j,0}^* = y_{j,0}^* \oplus H(j, \mathbf{q}_j)$ and $x_{j,1}^* = y_{j,1}^* \oplus H(j, \mathbf{q}_j \oplus \mathbf{s}^*)$, $1 \le j \le m$.
- Output whatever S^* outputs.

Correctness: It is easy to verify that the joint distribution of $\rho, \mathbf{s}^*, Q$, the values $(y_{j,0}^*, y_{j,1}^*)$ and all values of H queried by S^* in the ideal process is identical to the corresponding distribution in the real process. It remains to show that, conditioned on all the above values, the receiver's outputs x_{j,r_j}^* in the ideal process are distributed identically to these outputs in the real process. This follows from the way the output of R is defined in the Step 4 of the protocol and from the fact that (in the real process) $\mathbf{t}_j^* = \mathbf{q}_j \oplus (r_j \cdot \mathbf{s}^*)$. Note that the above simulation remains perfect even if the distinguisher makes an arbitrary number of calls to H. Thus we have:

Claim. The protocol is perfectly secure with respect to an arbitrary sender.

Simulating R. The semi-honest receiver R can be simulated as follows:

- Call TP with input $\mathbf{r}$. Let z_j denote the jth output received from TP.
- Run the protocol between R and S, substituting the values z_j for the known inputs x_{j,r_j} of S and the default value 0^ℓ for the unknown inputs $x_{j,1-r_j}$. Output the entire view of R.

It is clear from the descriptions of the protocol and the simulator that, conditioned on the event $\mathbf{s} \neq 0$, the simulated view is distributed *identically* to the receiver's view in the real process. Indeed, if $\mathbf{s} \neq 0$, the values of H used for masking the unknown inputs $x_{j,1-r_j}$ are uniformly random and independent of the receiver's view and of each other. Thus, the simulator's output is 2^{-k}-close to the real view. However, to make a meaningful security statement in the random oracle model, we must also allow the distinguisher to (adaptively) make additional calls to H, where the answers to these calls are provided by the same H that was used in the process (real or ideal) generating the distinguisher's input.

Now, if the distinguisher can "guess" the oracle query used in the real process to mask some secret $x_{j,1-r_j}$ which R is not supposed to learn, then it clearly wins (i.e., it can easily distinguish between any two values for this unknown secret). On the other hand, as long as it does not guess such a critical query, the masks remain random and independent given its view, and so indistinguishability is maintained. The crucial observation is that from the distinguisher's point of view, each of these m offending queries is (individually) distributed uniformly at random over a domain of size 2^k. This follows from the fact that the distinguisher has no information about s as long as it makes no offending query. Hence, the distinguisher can only win the above game with negligible probability. This is formalized by the following lemma.

Lemma 1. *Any distinguisher D which makes at most t calls to H can have at most a $(t+1) \cdot 2^{-k}$-advantage in distinguishing between the output of the real process and that of the ideal process.*

Proof. Define the *extended* real (resp., ideal) process to be the real (resp., ideal) process followed by the invocation of D on the output of the process. The output of the extended process includes the output of the original process along with the transcript of the oracle calls made by D. For each of the extended processes, define an *offending query* to be a call to H on input $(j, \mathbf{t}_j \oplus \mathbf{s})$ for some $1 \leq j \leq m$, and define B to be the (bad) event that an offending query is ever made by either R or D. It is easy to verify that, as long as no offending query is made, the outputs of the two extended processes are perfectly indistinguishable. Thus, the event B has the same probability in both extended processes, and the outputs of the two extended processes are identically distributed conditioned on B not occurring. It remains to show that $\Pr[B] \leq (t+1) \cdot 2^{-k}$. This, in turn, follows by noting that: (1) R makes an offending query only if $\mathbf{s} = 0$, and (2) as long as no offending query is made, D's view is completely independent of the value of $\mathbf{s}$. $\square$

Thus, we have:

Claim. As long as $m = 2^{o(k)}$, the protocol is statistically secure with respect to a semi-honest receiver and a polynomial-time distinguisher having access to the random oracle.

4 A Fully Secure Protocol

In this section we describe a variant of the protocol from Section 3 whose security also holds against a malicious receiver.

We begin by observing that the previous protocol is indeed insecure against a malicious R^*. Consider the following strategy for R^*. In Step 2, it chooses the ith pair of vectors sent to the OT_m^k primitive so that they differ only in their ith position.[7] For simplicity, assume without loss of generality that the ith bit of the first vector in the ith pair is 0. The above strategy guarantees that for $1 \leq j \leq k$ the vector $\mathbf{q}_j$ contains the bit s_j in its jth position and values known to R^* in the remaining positions. It follows that given an a-priori knowledge of $x_{j,0}$ the receiver can recover s_j by making two calls to H. This, in turn, implies that given the values of $x_{1,0}, \ldots, x_{k,0}$, the receiver can recover s and learn all $2m$ inputs of S.[8]

To foil this kind of attack, it suffices to ensure that the pairs of vectors sent by the receiver to the OT_m^k primitive are well-formed, i.e., correspond to some

[7] It is in fact possible for R^* to use this strategy in a completely undetectable way. Specifically, it can ensure that the m vectors received by S are uniformly random and independent.

[8] The security definition implies that the ideal process can properly simulate the real process given *any distribution* on the inputs. The above attack rules out proper simulation for an input distribution which fixes $x_{1,0}, \ldots, x_{k,0}$ and picks the remaining inputs uniformly at random.

valid choice of $\mathbf{r}$ and T. Indeed, the simulator of R from the previous section can be readily modified so that it simulates any "well-behaved" R' satisfying this requirement. To deal with an arbitrarily malicious R^* we employ a cut-and-choose technique. Let σ denote a statistical security parameter. The players engage in σ (parallel) executions of the previous protocol, where all inputs to these executions are picked randomly and independently of the actual inputs. Next, the sender challenges the receiver to reveal its private values for a random subset of $\sigma/2$ executions, and aborts if an inconsistency is found. This ensures S that except with $2^{-\Omega(\sigma)}$ probability, the remaining $\sigma/2$ executions contain at least one "good" execution where the receiver was well-behaved in the above sense. Finally, the remaining executions are combined as follows. Based on its actual selection bits, the receiver sends a correction bit for each of its $m\sigma/2$ random selections in the remaining executions, telling S whether or not to swap the corresponding pair of inputs. For each of its actual secrets $x_{j,b}$, the sender sends the exclusive-or of this secret with the $\sigma/2$ (random) inputs of the remaining executions which correspond to $x_{j,b}$ after performing the swaps indicated by the receiver. Having aligned all of the selected masks with the selected secrets, the receiver can now easily recover each selected secret x_{j,r_j}. This protocol is formally described in Fig. 2.

Note that the above protocol does not give a malicious S^* any advantage in guessing the inputs of R. Moreover, except with $2^{-\Omega(\sigma)}$ failure probability, security against a malicious R^* reduces to security against a well-behaved R'. A more detailed security analysis will be provided in the full version.

Efficiency. The modification described above increases the communication and time complexity of the original protocol by a factor of σ. The probability of R^* getting away with cheating is $2^{-\Omega(\sigma)}$.[9] In terms of round complexity, the protocol as described above adds a constant number of rounds to the original protocol.

Optimizations. We did not attempt to optimize the exact round complexity. A careful implementation, using commitments, can entirely eliminate the overhead to the round complexity of the basic protocol. The length of the OT seed can be shortened from σk to $O(k)$ via the use of a modified cut-and-choose approach, at the expense of a comparable increase in the length of the input to H (and while achieving roughly the same level of security). Further details are omitted from this abstract.

An on-line variant. In our default setting, we assume that the number m of desired OT is known in advance. However, in some scenarios it may be desirable to allow, following the initial seed of OT, an efficient generation of additional OT on the fly. While it is easy to obtain such an on-line variant for the basic protocol from Figure 1, this is not as obvious for the fully secure protocol. We note, however, that the modified cut-and-choose technique mentioned above can also be used to obtain an on-line variant of the fully secure protocol.

[9] In particular, the distance between the output of the simulator for R^* and the output of the real process increases by at most $2^{-\Omega(\sigma)}$.

Input of S: m pairs $(x_{j,0}, x_{j,1})$ of ℓ-bit strings, $1 \leq j \leq m$.
Input of R: m selection bits $\mathbf{r} = (r_1, \ldots, r_m)$.
Common input: security parameters k, σ.
Oracle: a random oracle $H : [\sigma] \times [m] \times \{0,1\}^k \rightarrow \{0,1\}^\ell$.
Cryptographic Primitive: An ideal $\text{OT}_m^{\sigma k}$ primitive.

1. For $1 \leq p \leq \sigma$, S initializes a random vector $\mathbf{s}^{(p)} \in \{0,1\}^k$ and m random pairs of ℓ-bit strings $(x_{j,0}^{(p)}, x_{j,1}^{(p)})$, $1 \leq j \leq m$. For $1 \leq p \leq \sigma$, R initializes a random vector $\mathbf{r}^{(p)} \in \{0,1\}^m$ and a random $m \times k$ bit matrix $T^{(p)}$.
2. The parties invoke the $\text{OT}_m^{\sigma k}$ primitive, where S acts as a receiver with inputs $\mathbf{s}^{(p)}$, $1 \leq p \leq \sigma$, and R as a sender with inputs $(\mathbf{t}^{(p),i}, \mathbf{r}^{(p)} \oplus \mathbf{t}^{(p),i})$, $1 \leq p \leq \sigma$, $1 \leq i \leq k$. Let $Q^{(p)}$ denote the pth $m \times k$ matrix of values received by S.
3. S picks a random subset $P \subset [\sigma]$ of size $\sigma/2$, and challenges R to reveal all values $\mathbf{r}^{(p)}$ and $T^{(p)}$ with $p \in P$. If the reply of R is not fully consistent with the values received in Step 2, S aborts.
4. For $1 \leq p \leq \sigma$ and $1 \leq j \leq m$, S sends $(y_{j,0}^{(p)}, y_{j,1}^{(p)})$ where
$$y_{j,b}^{(p)} = x_{j,b}^{(p)} \oplus H(p, j, \mathbf{q}_j^{(p)} \oplus b \cdot \mathbf{s}).$$
5. For all $1 \leq j \leq m$ and $p \notin P$, R sends a correction bit $c_j^{(p)} = r_j \oplus r_j^{(p)}$.
6. For $1 \leq j \leq m$ and $b \in \{0,1\}$, S sends
$$w_{j,b} = x_{j,b} \oplus \bigoplus_{p \notin P} x_{j, b \oplus c_j^{(p)}}^{(p)}.$$
7. For $1 \leq j \leq m$, R outputs
$$z_j = w_{j,r_j} \oplus \bigoplus_{p \notin P} (y_{j,r_j^{(p)}}^{(p)} \oplus H(p, j, \mathbf{t}_j^{(p)})).$$

Fig. 2. A fully secure protocol for extending oblivious transfers

5 On Instantiating the Random Oracle

In this section we define an explicit primitive which can be used to replace the random oracle in our constructions.

We say that $h : \{0,1\}^k \to \{0,1\}$ is *correlation robust* if for a random and independent choice of (polynomially many) strings $s, t_1, \ldots, t_m \in \{0,1\}^k$, the joint distribution $(h(t_1 \oplus s), \ldots, h(t_m \oplus s))$ is pseudo-random *given* $t_1, \ldots, t_m$. More precisely:

Definition 1 (Correlation robustness). *An efficiently computable function $h : \{0,1\}^* \to \{0,1\}$ is said to be* correlation robust *if for any polynomials $p(\cdot), q(\cdot)$ there exists a negligible function $\epsilon(\cdot)$ such that the following holds. For any positive integer k, circuit C of size $p(k)$, and $m \leq q(k)$*

$$\left|\Pr[C(t_1, \ldots, t_m, h(t_1 \oplus s), \ldots, h(t_m \oplus s)) = 1] - \Pr[C(U_{(k+1)m}) = 1]\right| \leq \epsilon(k),$$

where the probability is over the uniform and independent choices of $s, t_1, \ldots, t_m$ from $\{0,1\}^k$.

A correlation robust h can be used to obtain a (weak) pseudo-random function family defined by $f_s(t) = h(d \oplus t)$, and hence also a one-way function. However, we do not know whether correlation robustness can be based on one-way functions.

We now briefly discuss the application of correlation robustness to our problem. Consider the basic protocol from Fig. 1 restricted to $\ell = 1$, i.e., implementing *bit* oblivious transfers. Moreover, suppose that the first argument to the random oracle is omitted in this protocol (i.e., $H(\mathbf{u})$ is used instead of $H(j, \mathbf{u})$).[10] From the receiver's point of view, the value masking a forbidden input $x_{j,1-r_j}$ is $H(\mathbf{s} \oplus \mathbf{t}_j)$, where $\mathbf{s}$ is a uniformly random and *secret* k-bit string and $\mathbf{t}_j$ is a known k-bit string. The use of a correlation robust H suffices to ensure that the above m masks are *jointly* pseudo-random given the receiver's view.

It is also possible to modify the fully secure variant of this protocol so that its security can be based on correlation robustness. Such a modification requires the sender to randomize each evaluation point of H so as to prevent a malicious receiver from biasing the offsets t_j. Thus, we have:

Theorem 3. *Let k denote a computational security parameter. For any constant $c > 1$ it is possible to reduce k^c oblivious transfers to k oblivious transfers by making only a black-box use of a correlation robust function.*

6 Open problems

We have shown how to extend oblivious transfers in an efficient black-box manner given a *random* function, or given a *correlation robust* function. The question

[10] It is not hard to verify that as long as the receiver is honest, the protocol remains secure. The inclusion of j in the input to the random oracle slightly simplifies the analysis and is useful towards realizing the fully secure variant of this protocol.

whether it is possible to extend oblivious transfers in a black-box manner using a *one-way* function is still open. A negative answer (in the line of [22]) would further dilute our view of negative black-box reduction results as impossibility results.

A related question which may be of independent interest is that of better understanding our notion of correlation robustness. Its definition (while simple) appears to be somewhat arbitrary, and it is not clear whether similar variants, which still suffice for extending OT, are equivalent to our default notion. The questions of finding a "minimal" useful notion of correlation robustness and studying the relation between our notion and standard cryptographic primitives remain to be further studied.

7 Acknowledgments

We thank Moni Naor and Omer Reingold for useful comments.

References

1. Donald Beaver, *Correlated Pseudorandomness and the Complexity of Private Computations*, STOC 1996: 479-488
2. M. Bellare, A. Boldyreva and A. Palacio, *A Separation between the Random-Oracle Model and the Standard Model for a Hybrid Encryption Problem*, Electronic Colloquium on Computational Complexity (ECCC), 2003.
3. M. Bellare and P. Rogaway, *Random Oracles are Practical: a Paradigm for Designing Efficient Protocols*, Proc. of the 1st ACM Conference on Computer and Communications Security, pages 62–73, 1993. ACM press.
4. M. Bellare, J. Kilian and P. Rogaway, *The security of cipher block chaining*, Advances in Cryptology – CRYPTO '94, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 341–358.
5. M. Bellare and P. Rogaway, *Optimal asymmetric encryption*, Crypto '94, pages 91–111, 1994.
6. G. Brassard, C. Crépeau, and J.-M. Robert, *All-or-nothing disclosure of secrets*, Crypto '86, pp. 234-238, 1987.
7. R. Canetti, *Security and composition of multiparty cryptographic protocols*, *J. of Cryptology*, 13(1), 2000.
8. R. Canetti, G. Goldreich and S. Halevi, *The Random Oracle Methodology, Revisited (preliminary version)*, STOC: ACM Symposium on Theory of Computing, 1998.
9. C. Crépeau, *Equivalence between two flavors of oblivious transfers*, Crypto '87, p.350-354.
10. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *C. ACM*, 28:637-647, 1985.
11. S. Even and Y. Mansour, *A construction of a cipher from a single pseudorandom permutation*, Journal of Cryptology, vol. 10, no. 3, 151–162 (Summer 1997). Earlier version in Advances in Cryptology— ASIACRYPT '91. Lecture Notes in Computer Science, vol. 739, 210–224, Springer-Verlag (1992).
12. J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss and Rebecca N. Wright, *Secure Multiparty Computation of Approximations*, ICALP 2001: 927-938

13. A. Fiat, A. Shamir. *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*, Advances in Cryptology – CRYPTO '86, Lecture Notes in Computer Science, vol. 263, Springer-Verlag, 1986, pp. 186–194.
14. R. Gennaro and L. Trevisan, *Lower Bounds on the Efficiency of Generic Cryptographic Constructions*, IEEE Symposium on Foundations of Computer Science, 305-313, (2000)
15. Y. Gertner, S. Kannan, T. Malkin, O. Reingold and M. Viswanathan, *The Relationship between Public Key Encryption and Oblivious Transfer*, Proc. of the 41st Annual Symposium on Foundations of Computer Science (FOCS '00), 2000.
16. Y. Gertner, T. Malkin and O. Reingold, *On the Impossibility of Basing Trapdoor Functions on Trapdoor Predicates* Proc. of the 42st Annual Symposium on Foundations of Computer Science (FOCS '01) 2001.
17. N. Gilboa, *Two Party RSA Key Generation*, Proc. of CRYPTO 1999, pp. 116-129.
18. O. Goldreich, *Secure multi-party computation*, Available at http://philby.ucsb.edu/cryptolib/BOOKS, February 1999.
19. O. Goldreich, S. Micali and A. Wigderson, *Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design*, Proc. of the 27th FOCS, 1986, 174-187.
20. O. Goldreich and R. Vainish, *How to Solve Any Protocol problem - an Efficiency Improvement,* In proceedings, Advances in Cryptology: CRYPTO '87, 73–86, Springer (1988).
21. S. Goldwasser and Y. Tauman. *On the (In)security of the Fiat-Shamir Paradigm,* Electronic Colloquium on Computational Complexity (ECCC), 2003.
22. Impagliazzo, R. and S. Rudich, *Limits on the provable consequences of one-way permutations*, Proceedings of 21st Annual ACM Symposium on the Theory of Computing, 1989, pp. 44 – 61.
23. J. Kilian, *Founding Cryptography on Oblivious Transfer*, Proc of the 20th STOC, ACM, 1988, 20–29
24. J. Kilian and P. Rogaway, *How to protect DES against exhaustive key search*, Proceedings of Crypto '96, August 1996.
25. J.H. Kim, D.R. Simon, and P. Tetali. *Limits on the efficiency of one-way permutations- based hash functions*, Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, pages 535–542, 1999.
26. Y. Lindell and B. Pinkas, *Privacy Preserving Data Mining*, Journal of Cryptology 15(3): 177-206 (2002)
27. M. Naor and K. Nissim, *Communication preserving protocols for secure function evaluation*, STOC 2001: 590-599.
28. M. Naor and B. Pinkas, *Efficient oblivious transfer protocols*, SODA 2001.
29. M. Naor and B. Pinkas, *Oblivious Transfer and Polynomial Evaluation*, STOC: ACM Symposium on Theory of Computing (STOC), (1999).
30. M. Naor, B. Pinkas, and R. Sumner, *Privacy preserving auctions and mechanism design*, ACM Conference on Electronic Commerce (1999), pp. 129-139.
31. J. Nielsen, *Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case*, Crypto 2002, pp. 111-126.
32. M. O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.
33. V. Shoup, *OAEP Reconsidered*, Proc. of Crypto '01, pp. 239-259.
34. D. Simon, *Finding Collisions on a One-Way Street: Can Secure Hash Functions Be Based on General Assumptions?*, Proc. of EUROCRYPT '98, pp. 334-345.
35. E. Petrank and C. Rackoff, *Message Authentication of Unknown Variable Length Data*, Journal of Cryptology, Vol. 13, No. 3, pp. 315-338, 2000.
36. A. Yao, *Protocols for Secure Computations (Extended Abstract)* Proc. of FOCS 1982, 160-164.

A Secure Two-Party Computation

In this section we sketch the necessary definitions of secure two-party computation.[11] We refer the reader to, e.g., [7, 18] for more details.

A secure two-party computation task is specified by a two-party *functionality*, i.e., a function mapping a pair of inputs to a pair of outputs. A protocol is said to *securely* realize the given functionality if an adversary attacking a party in a real-life execution of the protocol can achieve no more than it could have achieved by attacking the same party in an ideal implementation which makes use of a trusted party. In particular, the adversary should not learn more about the input and output of the uncorrupted party than it must inevitably be able to learn. This is formalized by defining a *real process* and an *ideal process*, and requiring that the interaction of the adversary with the real process can be *simulated* in the ideal process.

The real process. In the real process, the two parties execute the given protocol on their respective inputs and a common security parameter k. A probabilistic polynomial-time *adversary*, who may corrupt one of the parties and observe all of its internal data, intervenes with the protocol's execution. In the default case of an *active* adversary, the adversary has full control over the messages sent by the corrupted party. In this case we say that the corrupted party is *malicious*. Also of interest is the case of a *passive* adversary, who may only try to deduce information by performing computations on observed data, but otherwise follows the protocol's instructions.[12] Such a corrupted party is referred to as being *semi-honest*, or *honest but curious*. At the end of the interaction, the adversary may output an arbitrary function of its view. The *output of the real process* (on the the given pair of initial inputs) is defined as the random variable containing the *concatenation* of the adversary's output and the output of the uncorrupted party.

The ideal process. In the ideal process, an incorruptible trusted party is employed for computing the given functionality. That is, the "protocol" in the ideal process instructs each party to send its input to the trusted party, who computes the functionality and sends to each party its output. The interaction of the adversary with the ideal process and the output of the ideal process are defined analogously to the above definitions for the real process. The adversary attacking the ideal process is referred to as a *simulator*.

Security. A protocol is said to securely realize the given functionality if, for any (efficient) adversary attacking the real process, there exists an (efficient) simulator attacking the same party in the ideal process, such that on any pair of

[11] The definitions we provide here apply to a simple stand-alone model assuming a non-adaptive adversary. However, the security of our reductions should hold for essentially any notion of security, including ones that support composition.

[12] A passive adversary may model situations where an attacker (e.g., a computer virus) does not want to be exposed by disrupting the normal behavior of the attacked machine.

INPUT OF S: n pairs of m-bit strings $(x_{i,0}, x_{i,1})$, $1 \leq i \leq n$.
INPUT OF R: n selection bits $\mathbf{r} = (r_1, \ldots, r_n)$.
COMMON INPUT: a security parameter k.
ORACLE: A PRG $G : \{0,1\}^k \to \{0,1\}^m$.
CRYPTOGRAPHIC PRIMITIVE: An ideal OT_k^n primitive.

1. S initializes n pairs of random k-bit seeds $(s_{i,0}, s_{i,1})$.
2. The parties invoke the OT_k^n primitive, where S acts as a sender with inputs $(s_{i,0}, s_{i,1})$, $1 \leq i \leq n$, and R as a receiver with input $\mathbf{r}$.
3. For $1 \leq i \leq n$, S sends $(y_{i,0}, y_{i,1})$, where $y_{i,b} = x_{i,b} \oplus G(s_{i,b})$.
4. For $1 \leq i \leq n$, R outputs $z_i = y_{i,r_i} \oplus G(s_{i,r_i})$.

Fig. 3. Reducing OT_m^n to OT_k^n

inputs, the outputs of the two processes are indistinguishable. The security is said to be perfect, statistical, or computational according to the type of indistinguishability achieved. In the latter two cases, indistinguishability is defined with respect to the security parameter k. The protocol is said to be secure against a semi-honest (resp., malicious) receiver, if the above security requirement holds with respect to a passive (resp., active) adversary corrupting the receiver. A similar definition applies to the sender. We note that in all cases the protocol is required to compute the given functionality if no party is corrupted.

B Reducing OT_m^n to OT_k^n

Oblivious transfer of long strings can be efficiently reduced to oblivious transfer of shorter strings using any pseudo-random generator. The reduction is formally described in Fig. 3.

The security of this reduction is straightforward to prove. Its complexity overhead is dominated by the cost of generating $2m$ pseudo-random bits. By combining Step 3 with Step 2, the protocol's round complexity can be made the same as that of OT_k^n. Finally, we note that the use of the PRG G can be emulated by standard means using calls to the random oracle H.