OT-Combiners via Secure Computation

Danny Harnik[1], Yuval Ishai[2], Eyal Kushilevitz[3], and Jesper Buus Nielsen[4]

[1] IBM Research, Haifa, Israel. danny.harnik@gmail.com.*
[2] Technion, Israel and UCLA, USA. yuvali@cs.technion.ac.il.**
[3] Technion, Israel. eyalk@cs.technion.ac.il.***
[4] University of Aarhus, Denmark. buus@daimi.au.dk.†

Abstract. An *OT-combiner* implements a secure oblivious transfer (OT) protocol using oracle access to n OT-candidates of which at most t may be faulty. We introduce a new general approach for combining OTs by making a simple and modular use of protocols for secure computation. Specifically, we obtain an OT-combiner from any instantiation of the following two ingredients: (1) a t-secure n-party protocol for the OT functionality, in a network consisting of secure point-to-point channels and a broadcast primitive; and (2) a secure two-party protocol for a functionality determined by the former multiparty protocol, in a network consisting of a single OT-channel. Our approach applies both to the "semi-honest" and the "malicious" models of secure computation, yielding the corresponding types of OT-combiners.

Instantiating our general approach with secure computation protocols from the literature, we conceptually simplify, strengthen the security, and improve the efficiency of previous OT-combiners. In particular, we obtain the first *constant-rate* OT-combiners in which the number of secure OTs being produced is a constant fraction of the total number of calls to the OT-candidates, while still tolerating a constant fraction of faulty candidates ($t = \Omega(n)$). Previous OT-combiners required either $\omega(n)$ or poly(k) calls to the n candidates, where k is a security parameter, and produced only a single secure OT.

We demonstrate the usefulness of the latter result by presenting several applications that are of independent interest. These include:

Constant-rate OTs from a noisy channel. We implement n instances of a standard $\binom{2}{1}$-OT by communicating just $O(n)$ bits over a noisy channel (binary symmetric channel). Our reduction provides unconditional security in the semi-honest model. Previous reductions of this type required the use of $\Omega(kn)$ noisy bits.

Better amortized generation of OTs. We show that, following an initial "seed" of $O(k)$ OTs, each additional OT can be generated by only computing and communicating a *constant* number of outputs of a

* Research conducted while at the Technion. Supported by grant 1310/06 from the Israel Science Foundation and a fellowship from the Lady Davis Foundation.

** Supported by ISF grant 1310/06, BSF grant 2004361, and NSF grants 0205594, 0430254, 0456717, 0627781, 0716835, 0716389.

*** Supported by ISF grant 1310/06 and BSF grant 2002354.

† Funded by the Danish Agency for Science, Technology and Innovation.

cryptographic hash function. This improves over a protocol of Ishai *et al.* (Crypto 2003), which obtained similar efficiency in the semi-honest model but required $\Omega(k)$ applications of the hash function for generating each OT in the malicious model.

1 Introduction

Secure Multiparty Computation (MPC) protocols allow a number of mutually distrusting parties to jointly evaluate functions over their local inputs without compromising the privacy of these inputs or the correctness of the output. (In the following we will also refer to functions over distributed inputs as "functionalities", capturing the general case where different parties may obtain distinct, and possibly randomized, outputs.) If a majority of the parties involved are honest, then there are "information-theoretic" solutions for this general task, requiring no computational assumptions [3, 7]. On the other hand, if an honest majority is not guaranteed then, by [10, 32], secure computation protocols for most functionalities imply the existence of *oblivious transfer* (OT) [37, 19, 39] — a secure two-party protocol for a simple functionality which allows a receiver to select one of two strings held by a sender. In an OT protocol the receiver learns the chosen string but no information about the other string, while the sender learns nothing about the receiver's selection. (By default, we use the term OT to refer to the basic *bit OT* primitive, where each string held by the sender consists of a single bit. OT of ℓ-bit strings can be implemented by making $O(\ell)$ calls to the basic OT primitive [4, 5].) OT has proved to be a very useful building block in cryptographic protocols. Most notably, OT can serve as a building block for general secure two-party and multi-party protocols that tolerate an arbitrary number of corrupted parties [41, 23, 22, 31, 21, 33].

1.1 Combiners and OT-Combiners

Often in cryptography there is uncertainty regarding the security of a construction (e.g., because of the reliance on unproven assumptions or placing too much trust in third parties). In such cases, it is handy to use a *combiner* for the underlying cryptographic task. An (m, n)-combiner (sometimes called a robust or tolerant combiner) is a method of taking n *candidates* for a cryptographic primitive and combining them into a single primitive that is secure as long as at least m of the n candidates were indeed secure. Combiners have been used implicitly in many cryptographic constructions as means of enhancing security and have recently been studied explicitly (initially in [26, 25]).

In this paper, we focus on OT-combiners and their applications. The possibility of realizing combiners for OT or equivalent primitives has been investigated in [25, 35, 36, 40]. Constructions of OT-combiners were given for the case that a *majority* of the OT-candidates are good [25, 40]. These combiners are based on a technique of Damgård *et al.* [18] for reducing errors in weak versions of OT. On the other hand, there is a strong indication that there are no (black-box)

OT-combiners if half of the candidates may be faulty [25]. We refer to $\frac{n-m}{n}$ as the *tolerance ratio* of the combiner. Thus, the results indicate that the tolerance ratio of an OT-combiner should be smaller than $\frac{1}{2}$.

A problem with OT-combiners as above is that, by definition, they are quite wasteful. One needs to invoke n OT-candidates (at least $m > \frac{n}{2}$ of which are good) in order to produce just a single instance of secure OT. Even worse, the known OT-combiners make a large number of *calls* to each candidate in order to produce this single secure OT. Thus, a desirable goal is to reduce the number of candidate calls made in order to produce each good OT. We refer to the latter quantity as the *production rate* of a combiner (or simply its *rate*).

The OT-combiners based on the technique of [18] have a production rate of $\Theta(k^2 n^4)$ (where k is a security parameter) for a majority of good candidates (see [40]). This rate can be improved to $\Theta(k^2)$ when assuming a constant tolerance ratio; i.e. $m > (\frac{1}{2} + \delta)n$, for a constant $\delta > 0$. Another downside of this construction is that it does not provide security when the identity of the faulty OTs can be determined adaptively.[1] A different approach to OT-combiners was taken by [36] (see also [2]) and requires $\Omega(n \log n)$ calls to the OT-candidates.

1.2 Our Results

We introduce a new general approach for constructing OT-combiners by making use of protocols for secure multiparty computation. Our approach follows a recent paradigm suggested by Ishai *et al.* [30] of employing secure *multiparty* protocols for the construction of secure *two-party* protocols. This allows us to benefit from the wide range of techniques that have been developed in the study of secure multiparty computation, obtaining conceptually simpler and more efficient OT-combiners.

More concretely, we show how to obtain an OT-combiner from any instantiation of the following two ingredients:

1. A t-secure n-party protocol for the OT functionality, in a network consisting of secure point-to-point channels and a broadcast primitive.[2] For instance, one could use here the (unconditionally secure) general-purpose protocols of [3, 7, 38, 12].
2. A secure two-party protocol for a functionality determined by the former multiparty protocol, in a network consisting of a single OT-channel. For instance, one can use here the general-purpose unconditionally secure protocols of [23, 31, 22, 21] or the computationally secure protocols of [41, 33].

[1] Adaptive security is not always required for combiners, however, at times it is crucial. For example, consider a setting where the OT-candidates are carried out simply by using third parties. A candidate is insecure if the corresponding third party is corrupted. In such a setting, an adversary can potentially corrupt a third party adaptively, during the execution of the combiner.

[2] This refers to a model in which the sender and the receiver are not considered to be among the n parties, but may each be corrupted by the adversary. Alternatively, one can use any $(t + 1)$-secure $(n + 2)$-party protocol in the standard MPC model.

Our approach applies both to the "semi-honest" and to the "malicious" models of secure computation, yielding the corresponding types of OT-combiners. (In the semi-honest model the sender and the receiver follow the protocol as prescribed, while in the malicious model they may deviate from it.) In contrast to previous OT-combiners in the malicious model, the OT instances produced by our combiners are provably secure (in the malicious model) under standard simulation-based definitions, and can resist adaptive corruptions of candidates and parties.

By instantiating our general approach with efficient MPC protocols from the literature and by giving up just a constant fraction in the tolerance threshold, we get combiners with a constant production rate. In particular:

- There exists an OT-combiner in the *semi-honest* model with *constant production rate* and *constant tolerance ratio*. The combiner makes $O(1)$ calls to each of the n OT-candidates.
- There exists an OT-combiner in the *malicious* model with *constant production rate* and *constant tolerance ratio*. The combiner makes $s \leq \text{poly}(k)$ calls to each of the n OT-candidates (and generates $\Omega(ns)$ good OT calls). This combiner applies to *string OT* (rather than bit-OT) and requires additional calls to a one-way function.

Both results hold even if the bad candidates are chosen adaptively. Recall that previous OT-combiners required either $\omega(n)$ (with adaptive security) or $\text{poly}(k)$ (without adaptive security) calls to the OT-candidates in order to produce just a single instance of secure OT.

Techniques. The high-level idea behind our approach is to let the sender S and receiver R invoke the given MPC protocol between themselves and n additional "imaginary" parties called servers, where each server is jointly simulated by S and R using the given two-party protocol applied on top of a corresponding OT-candidate. Different instantiations of the underlying multiparty and two-party protocols yield different OT-combiners.

Our constant-rate combiners rely on MPC protocols in which the (amortized) communication complexity per gate of the circuit being evaluated is bounded by a constant, independently of the number of parties. For the type of functionalities we consider in this work, such protocols can be obtained by combining a protocol from [16] with secret-sharing schemes based on algebraic geometric codes or random linear codes [8, 9] (see [30]). The protocol from [16], in turn, uses techniques from [3, 20, 27]. We also rely on OT-based secure two-party computation protocols in which the number of OT calls is a constant multiple of the input length. Such a protocol with a simulation-based proof of security was recently given in [33].

1.3 Applications

We demonstrate the usefulness of our constant-rate combiners by presenting several applications that are of independent interest.

Constant-rate OTs from a noisy channel. Crépeau and Kilian [14] demonstrated that two parties can implement an unconditionally secure OT by communicating over a binary symmetric channel (BSC) with some constant crossover probability $0 < p < \frac{1}{2}$. One can view such a channel as a secure implementation of a randomized functionality in which the receiver gets the sender's input bit with probability $1 - p$ and its negation with probability p. Thus, the result from [14] shows that this functionality is equivalent to OT. Unfortunately, the reduction from [14] is quite inefficient; its efficiency was later improved by Crépeau [13], but even this reduction requires $\Omega(k)$ noisy bits for producing a single OT, even in the semi-honest model.

Using our constant-rate combiners in the semi-honest model, we get n instances of $\binom{2}{1}$-OT by communicating just $O(n)$ bits over the noisy channel. Thus, the amortized cost of generating each OT call is just a *constant* number of calls to the noisy channel. Our reduction provides unconditional security in the semi-honest model and has error probability that vanishes exponentially with n. Combined with the OT-based secure computation protocol of [23, 22, 21], it implies that two parties can securely evaluate an arbitrary circuit of size s (with statistical security in the semi-honest model) by communicating only $O(s)$ bits over a noisy channel. It seems likely that our approach can be extended to yield similar results for the malicious model as well as for more general noise models and other probabilistic functionalities. We leave such extensions to future work.

Extending OTs efficiently in the malicious model. Current implementations of OT are quite expensive in practice, and thus form the efficiency bottleneck in protocols that make a heavy use of OTs. This state of affairs is backed up by the result of Impagliazzo and Rudich [28], which implies that there is no black-box construction of OT from one-way functions. As a next to best solution, Beaver [1] demonstrated how one can use just k OT calls (k being the security parameter) and extend them to polynomially many OT calls solely by adding calls to a one-way function. Beaver's protocol makes a non-black-box use of the underlying one-way function and is therefore considered inefficient in practice. Ishai *et al.* [29] gave an alternative construction that extends k OT calls to an essentially unbounded number of OT calls by making an additional black-box use of a cryptographic hash function. This protocol is highly efficient and has an amortized cost of computing and communicating just *two* outputs of the hash function for each produced OT. This approach can be viewed as the OT analogue of hybrid encryption, where an expensive asymmetric cryptosystem is used to encrypt a short secret key, allowing the bulk of the data to be encrypted efficiently using a symmetric encryption scheme.

Unfortunately, the efficient protocol of [29] applies only in the semi-honest model. In order to achieve security in the malicious model, a modified protocol is proposed based on a "cut-and-choose" approach. However, this approach increases the complexity of OT generation by a multiplicative factor of at least $\Omega(k)$. In this paper, we utilize our constant-rate combiners to get OT extension in the malicious model that requires only a *constant* number of outputs of the hash function to be computed and communicated for each generated OT.

A first solution to this is by using a cut-and-choose approach similar to the one used in [29]. Namely, the semi-honest protocol of [29] is invoked $O(k)$ independent times on random inputs, and half of these invocations are "opened" to allow each party to verify that the other party followed the protocol's instructions. The unopened invocations have the guarantee that with overwhelming probability, a big majority of them generated secure OTs. This is exactly the setting required to apply our combiners. However, in this solution the seed of OT calls used grows substantially, which is undesirable.

Next, we develop a new solution that is not based on cut-and-choose. Instead, just a single instance of the semi-honest protocol is run, and a simple test is added communicating a constant number of hash values per produced OT. This test guarantees that with overwhelming probability, all but k of the produced OTs were secure. This allows to generate $(1 + \delta)k$ OT-candidates of which at most k are insecure, which again gives a big majority of secure OTs. As in [29], the reduction only makes a black-box use of the cryptographic hash function.

Reducing the number of OT channels in MPC. Consider an MPC protocol with security against a dishonest majority in a network of n parties. How many *OT channels* are required to allow such a non-trivial computation? An OT channel is a line over which two parties can carry out an unbounded number of OT calls. Do all pairs of parties require their own separate OT-channel? Harnik et al. [24] show that the answer is negative – if the number of corrupted parties is bounded by $t < (1 - \delta)n$ (for a constant δ) then $O(n)$ channels are sufficient. Namely, OT calls can be executed between every two parties using just OT calls on the existing $O(n)$ channels. However, in order to generate one OT call between a pair with no channel, the construction of [24] makes many OT calls over the existing channels: it first generates n candidates for OT (using just $O(1)$ OT calls to generate each candidate) and then it runs an OT-combiner on the n candidates. Using our new constant-rate combiners, we get the following result: the amortized cost of generating an OT call between two parties with no OT channel is $O(1)$ OT calls over the existing $O(n)$ channels.

Organization. In Section 2, we define OT-combiners. Section 3 describes our general approach for obtaining OT-combiners via secure computation and its instantiation for obtaining constant-rate combiners in the semi-honest model. Section 4 deals with the malicious model. The applications are described in Sections 5 (OT from noisy channels) and 6 (extending OTs efficiently). Due to space limitations, some of the details are deferred to the full version.

2 Definition of OT-Combiners

A combiner (see [26, 25]) is given n implementation candidates for a cryptographic primitive and combines them into a single implementation that is secure if at least m of the original n candidates were indeed secure. Our study of combiners for OT follows this goal with an additional feature: we want the combiner to output many secure instances of OT rather than just one. This is desirable for efficiency reasons; indeed, invoking n OT-candidates and receiving just a single OT call in return seems quite wasteful. To accommodate this, we define the multi-OT functionality OT^ℓ in a straightforward manner (the sender holds ℓ pairs of secrets, the receiver holds ℓ choice bits and the receiver learns the secrets of his choice).

Our OT-combiners thus take several candidates for a secure OT protocol and produce a protocol for the OT^ℓ functionality. In general, the OT-candidates can be given in any representation, such as a code, or via a black-box access to a next message oracle. Our combiners work using such black-box access to the candidates. Accordingly, the definition we provide is that of a black-box combiner. For more comprehensive definitions of combiners, see [25].

We assume that the candidates are efficient (polynomial-time) algorithms. This guarantees that an efficient black-box combiner (counting each oracle call to a candidate as a single running step) remains efficient for any instantiation of the candidate.

When considering the functionality of the OT-candidates being combined, one should distinguish between two cases: (1) bad OT-candidates can compromise the privacy of the inputs but are guaranteed to have the correct functionality when executed honestly (namely, the receiver always ends up with the correct output); and (2) bad OT-candidates may produce arbitrary outputs. Combiners for the latter case are called error-tolerant combiners [36]. In the semi-honest model, we assume by default that bad candidates have the correct functionality, but our solutions can be easily extended to achieve error-tolerance with almost no loss of efficiency. In the malicious model, error-tolerance is always required; moreover, the functionality of each call to a bad candidate can be adaptively determined by the adversary during the execution of the combiner.

Definition 1 (OT-Combiner) *Let $OT_1, \ldots, OT_n$ be candidates for implementing OT. An $(m, n; \ell, s)$-*OT-Combiner *is an efficient two-party protocol with oracle access to the candidates such that: (1) If at least m of the n candidates securely compute the OT^1 functionality then the combiner securely computes the OT^ℓ functionality (where security is defined using a simulation-based definition, as in [6, 21]); and (2) The combiner runs in polynomial time and makes a total of s calls to the candidates.*

The tolerance ratio *of the combiner is defined as $\mu = \frac{m-n}{n}$. The* production rate *(or simply the* rate*) of the combiner is defined as $\rho = \frac{\ell}{s}$. At times we omit the parameters s and ℓ and write "(m, n)-OT-combiner".*

The above definition views the number of candidates n as a constant. However, it is often useful to view the parameters of a combiner as functions of the security parameter k. (This is the case for the applications described in Sections 5,6.) The above definition can be naturally extended to this more general case.

We will sometimes refer to combiners that have *unconditional* (perfect or statistical) security. In such cases the above definition needs to be modified, since the candidates cannot be unconditionally secure. To this end, it is convenient to use the stronger notion of *third party black-box* combiners [25]. Such combiners are defined by viewing each candidate as a distinct external party that receives

OT inputs from the sender and the receiver and sends the OT output to the receiver. A bad candidate is modelled by an external party that reveals both inputs to the adversary and (in the error-tolerant case) allows the adversary to control its output. All of our combiners satisfy this stronger definition with either perfect, statistical, or computational security.

3 OT-Combiners in the Semi-Honest Model

In this section, we introduce our basic technique for obtaining combiners from protocols for secure computation. We begin by considering the semi-honest model and later (in Section 4) extend our results to the malicious model.

In the course of describing the OT-combiner, we define two intermediate models. The first is a tweak on the standard multiparty model which divides the parties into clients and servers, where the clients are the only parties to hold inputs and to receive outputs and the servers just assist in the computation. Such a variant of secure MPC setting was considered, e.g. in [11, 16]. In our setting, there are only two clients, sender S and receiver R. In addition, there are n servers P_i that may aid the clients in the computation. However, up to t of the servers may be corrupted. Formally:

Definition 2 (Clients-Servers Model) *The network consists of $n+2$ parties: two* clients*, S and R, and n* servers *$P_1, \ldots, P_n$. There are secure channels between every two parties in the network. In the malicious model, we will also allow broadcast as an atomic primitive.*
FUNCTIONALITY: *f takes inputs from S and R and gives output to R.*[3]
ADVERSARIAL CORRUPTIONS: *The adversary may corrupt at most one of the clients S and R and at most t of the n servers. We refer to a protocol that is secure against such an adversary as a t-*secure protocol in the clients-servers model. *We consider adaptive adversaries by default; namely, we allow the adversary to decide which parties to corrupt during the execution of the protocol.*

The above model can be viewed as a refinement of the standard model for secure computation: every $(t+1)$-secure $(n+2)$-party protocol for f in the standard model is also a t-secure protocol for f in the clients-servers model.

In the second intermediate model we use, each server P_i is replaced by a pair of parties (S_i, R_i) that are connected by an OT channel. We call this the split-servers model. The intuition is that, at the end, we will have a two-party protocol where one party controls all R parties and the other controls all S parties.

Definition 3 (Split-Servers Model) *The network consists of $2n+2$ parties: two clients, S and R, and n* pairs *of parties $(S_1, R_1), \ldots, (S_n, R_n)$. There is a secure channel between every two parties in the network (as well as a broadcast*

[3] Our approach can be easily generalized to the case where f gives outputs to both S and R. However, in the malicious model it is impossible to guarantee *fairness* in this case.

channel in the malicious model). In addition, there is a distinct OT channel between each pair (S_i, R_i)*.*
FUNCTIONALITY: f *takes inputs from* S *and* R *and gives output to* R*.*
ADVERSARIAL CORRUPTIONS: *The adversary has two possible corruption patterns: either (i) it corrupts the parties* $S, S_1, ..., S_n$ *and at most* t *of the* R_i*'s; or (ii) it corrupts the parties* $R, R_1, ..., R_n$ *and at most* t *of the* S_i*'s. We refer to a protocol that is secure against such an adversary as a* t-secure protocol in the split-servers model*. Again, we allow for adaptive adversaries.*

Our general construction employs two types of secure computation protocols: (1) Π_{MPC} is a t-secure *multi*party protocol in the clients-servers model. For $t < n/2$, every f admits such a protocol with perfect (resp., statistical) security against semi-honest (resp., malicious) adversaries [3, 38]. (2) Π_{2party} is a secure 2-party protocol in the OT-hybrid model (i.e., using an ideal OT channel). Every f admits such a protocol with perfect (resp., statistical) security against semi-honest (resp., malicious) adversaries [22, 31].

In our combiners, Π_{MPC} will always compute the functionality OT^{ℓ}.[4] On the other hand, we will need to employ protocols of type Π_{2party} for different functionalities. To simplify notation, we always use the notation Π_{2party} and make the actual functionality clear from the context.

Lemma 1. *Let* f *be a functionality taking inputs from* S *and* R *and returning output to* R*. There is a compiler that transforms any* t*-secure protocol* Π_{MPC} *for* f *in the semi-honest clients-servers model into a* t*-secure protocol* Π_{split} *for* f *in the semi-honest split-servers model. As a building block, the compiler requires a secure two-party protocol* Π_{2party} *for general functionalities in the semi-honest OT-hybrid model. If both* Π_{MPC} *and* Π_{2party} *are perfectly or statistically secure then so is* Π_{split}*.*

Proof: The idea is to distribute the local view of each server P_i in Π_{MPC} between the corresponding pair of parties S_i, R_i in Π_{split} using additive secret sharing. Thus, only an adversary corrupting both S_i and R_i can learn the view of P_i.

In the initialization stage of Π_{MPC}, the view of each client (S or R) contains its private input and its local randomness, while the view of each server P_i contains only its local randomness. To initialize the corresponding state in Π_{split}, let S and R remain as before and split the view of each P_i between S_i and R_i, by having each hold local random bits which together form an additive sharing of the randomness of P_i.

A typical intermediate step in the protocol Π_{MPC} is of the following form: Server P_i with local view v_{P_i} computes a function $m = \pi_{i,j}(v_{P_i})$ and sends the message m to server P_j. This step is simulated in Π_{split} by an interactive two-party protocol between S_i and R_i. Using the OT channel between them, S_i and R_i execute protocol Π_{2party} on the randomized functionality whose inputs are

[4] One can also consider cross-primitive combiners (see [34]), where the combiner implements a different functionality than the candidates. In such a case, the combiner's functionality will be computed by Π_{MPC}.

shares v_{S_i} and v_{R_i} of a view v_{P_i}, and whose outputs are random values m_{S_i} and m_{R_i} under the restriction that $m_{S_i} \oplus m_{R_i} = \pi_{i,j}(v_{S_i} \oplus v_{R_i})$. After the secure two-party protocol is executed, S_i sends m_{S_i} to S_j and R_i sends m_{R_i} to R_j. The recipients S_j and R_j append the new message to their share of the view v_{P_j}.

Another possible step in Π_{MPC} involves one or two of the clients, either as the party generating a new message m or as the receiving party. In case the recipient is one of the clients S or R, then both m_{S_i} and m_{R_i} are sent to this party. If a client (S or R) is generating the message m, then it computes m as in Π_{MPC} (no two-party protocol is needed in this case) and sends a random sharing of m to S_j and R_j, respectively. If both the sender and receiver of m are the clients then m is sent unchanged.

The new protocol produces a correct output since at each step the sum of the shares held by S_i and R_i is exactly the view of P_i, and thus the execution follows the protocol Π_{MPC} accurately. The proof of security hinges on the fact that if the i^{th} pair is not corrupted (i.e., either S_i or R_i is uncorrupted), then the view of the simulated server P_i remains hidden from the adversary. Therefore, this view may be simulated in the same manner as in the original clients-servers model protocol Π_{MPC} (in the case that server P_i was not corrupted). On the other hand, if the i^{th} pair is corrupted, then this corresponds to a corruption of P_i by the adversary. Further details are deferred to the full version. ∎

Lemma 2. *Given any protocol Π_{split} for the functionality OT^ℓ which is t-secure in the semi-honest split-servers model, one can construct (in a black-box way) an $(n-t,n)$-OT-combiner in the semi-honest model.*

Proof: We describe a two-party combiner with sender S' and receiver R' based on the split-server protocol Π_{split} with clients S and R and parties $S_1,\ldots,S_n$, $R_1,\ldots,R_n$. The combiner protocol is a straightforward simulation of Π_{split} where S' simulates the S-parties (i.e., $S, S_1,\ldots,S_n$) and R' simulates the R-parties $(R, R_1,\ldots,R_n)$. The simulation follows the protocol Π_{split} with the exception that, for every $i \in [n]$, all OT-calls between S_i and R_i are implemented using the candidate OT_i. Naturally, messages between the S-parties do not have to actually be sent as they are all simulated by S' (and similarly for the R-parties). Clearly, in a semi-honest environment, the combiner described is an execution of protocol Π_{split} and therefore it indeed implements the OT^ℓ functionality. Intuitively, security follows from the fact that for every bad candidate, the worst-case scenario is that the full view of the opposite party is revealed. But, as long as the adversary sees no more than t such views, security follows from the t-security of Π_{split}. A formal proof, deferred to the full version, uses a simulator for Π_{split} to obtain a simulator for the combiner. ∎

A first corollary of the above strategy is the existence of OT-combiners with a majority of good candidates. Such a result was already known (see [25, 35, 40]), based on a different approach stemming from the techniques of [18].

Corollary 4 *For any m and n such that $m > n/2$, there exists an (m, n)-OT-combiner in the semi-honest model. Furthermore, such a combiner can be perfectly secure.*

Proof: Use, for example, the protocol of [3] to implement Π_{MPC} with $t = n - m$ and the protocol of [23, 21] to implement Π_{2party} in the semi-honest model. By Lemmas 1 and 2, this implies the desired OT-combiner. ∎

3.1 Constant-Rate OT-Combiners in the Semi-Honest Model

We turn to optimizing the efficiency of the combiner described above. Its efficiency is inherited from the underlying protocols Π_{MPC} and Π_{2party}. For different purposes, one may choose different protocols Π_{MPC} and Π_{2party} with suitable properties. The key parameter that we investigate is the total number of calls to the OT-candidates. In our framework, calls to the candidates happen as part of executions of the protocol Π_{2party}, where each step of a server in the protocol Π_{MPC} requires an invocation of Π_{2party}. Therefore, the total number of calls is a function of the number of steps in Π_{MPC}, the complexity of the local computation in each such step, and the OT complexity of Π_{2party}.

A natural implementation of our combiner, using [3, 23], has a polynomial rate and threshold $m = \lceil \frac{n+1}{2} \rceil$. More precisely, in order to compute ℓ OTs, the clients should compute a simple constant-size circuit on ℓ independent pairs of inputs. Using the BGW technique [3], the local computation required by each server for each OT is dominated by multiplying two elements from a field of size at least n. Simulating this action by a split server, requires a secure 2-party computation of such a functionality in the OT-hybrid model, which involves $\Omega(\log n)$ OT-calls. The overall OT complexity is therefore $s = \Omega(n\ell \log n)$.

In the following we show that, by a careful instantiation of Π_{MPC}, we can obtain a constant rate at the price of a slightly sub-optimal (yet still constant) tolerance ratio. The following two techniques allow this improvement:

- Using a generalization of Shamir's secret sharing scheme [20], which packs ℓ secrets into a single polynomial, one can run a joint computation for all ℓ inputs by sending just a constant number of field elements to each server. As a result of packing ℓ secrets into a single polynomial, the security threshold decreases from $t = \lfloor \frac{n-1}{2} \rfloor$ to $t' = t - \ell + 1$. Thus, letting ℓ be a sufficiently small constant fraction of n, the tolerance ratio remains constant.
- Using the techniques of [8, 9], one can run all operations over a constant size field. This technique further deteriorates the security threshold to $t'' = t' - \delta n$, for some constant $\delta > 0$ that tends to 0 as the field size grows.

Combining the above two techniques, one gets a protocol Π_{MPC} in which each server performs a constant amount of work and ℓ, the number of OTs being computed, is a constant fraction of the number of servers. When compiling such a protocol to the split-servers model (and subsequently to the combiner) we get a constant number of OT-calls per split server. An appropriate choice of parameters (say, $\ell = 0.2n$ and $\delta = 0.2$) thus yields the following:

Theorem 5. *There is an OT-combiner with constant production rate and constant tolerance ratio in the semi-honest model. The combiner makes a constant number of calls to each OT-candidate.*

We end this section by noting that one can get an error-tolerant version of Theorem 5 by letting the clients in Π_{MPC} apply error-correction to the final n-tuple of field elements received from the n servers. This requires the underlying secret sharing scheme to be based on efficiently decodable codes (such as AG codes [8]). Furthermore, the (constant) fractional security threshold of Π_{MPC} should be further decreased in order to provide the redundancy required for error-correction. This yields the following:

Theorem 6. *There is an* error-tolerant *OT-combiner with constant production rate and constant tolerance ratio in the semi-honest model. The combiner makes a constant number of calls to each OT-candidate.*

4 OT-Combiners in the Malicious Model

Our constructions of combiners in the malicious model follow the same outline as in the semi-honest model. Namely, the combiner is a composition of two types of secure protocols, one in the two-party setting (with an OT channel) and one in the multiparty setting. Naturally, this time the components must be secure against malicious adversaries (and thus are inherently more complex). In addition, we must incorporate a mechanism to assure authenticity of intermediate shares supplied by the split servers.

Lemma 3. *Let f be a functionality taking inputs from S and R and returning output to R. There is a compiler that transforms any t-secure protocol Π_{MPC} for f in the malicious clients-servers model into a t-secure protocol Π_{split} for f in the malicious split-servers model. As a building block, the compiler requires a two-party protocol Π_{2party} for general functionalities in the malicious OT-hybrid model. If both Π_{MPC} and Π_{2party} are statistically secure then so is Π_{split}.*

The same general idea as in the semi-honest model applies here as well with one notable addition. Recall that in the general framework each server P_i in the protocol Π_{MPC} is simulated by two parties S_i, R_i that hold an additive secret sharing of P_i's view. The problem is that, in the split-servers model, the adversary may corrupt all of the parties on one side (either all of the R_i's or all of the S_i's). While the adversary has no information about the view of a server P_i unless both S_i and R_i are corrupted, it can still change the outgoing messages from this server. Namely, the adversary needs only to change the outgoing message of one side (say S_i) in order to change the effective outgoing message of server P_i in Π_{MPC} (recall that two messages in Π_{split} correspond to a single message in Π_{MPC}). To overcome this problem, we replace the use of standard additive secret sharing by *authenticated* secret sharing. Namely, each S_i gets, in addition to its additive share v_{S_i}, a signature on this share using a private key known to R_i (and vice versa). For the signature primitive it suffices to use a one-time MAC, which can be implemented with unconditional security using pairwise independent hash functions.

The two-party functionality realized by Π_{2party} takes the additive shares of the view together with the signatures and the keys as inputs. It then verifies that the signatures are valid; if this verification fails it sends an "abort" message to both parties. (Any party receiving an abort message broadcasts it to all parties and aborts; this cannot violate the fairness of Π_{split}, since there is only one party receiving an output.) The functionality returns a similar authenticated secret sharing of the next message sent from P_i to P_j in Π_{MPC}.

Note that, in the malicious model, Π_{2party} cannot achieve fairness. As before, if some honest party aborts it causes all honest parties to abort. Finally, if Π_{MPC} employs a broadcast primitive, then each message broadcasted by P_i can be naturally emulated in Π_{split} as follows. First, S_i and R_i broadcast their authenticated shares of the message. Then, each of them uses its secret key to verify the shares broadcasted by the other party, and broadcasts an abort message if the verification fails.

Lemma 4. *Given any protocol Π_{split} for the functionality OT^ℓ which is t-secure in the malicious split-servers model, one can construct (in a black-box way) an $(n-t,n)$-OT-combiner in the malicious model.*

The construction is essentially the same as in the semi-honest model. The only changes that need to be made involve broadcast messages and handling aborting parties. A broadcast message by S or S_i is emulated by simply having the sender S' of the combiner send this message to R' (and vice versa). In case some party S or S_i (resp., R or R_i) in Π_{split} aborts, then S' (resp., R') in the combiner aborts as well. A simulation-based security proof is deferred to the full version.

Corollary 7 *For any m and n such that $m > n/2$, there exists an (m,n)-OT-combiner in the malicious model. Furthermore, such a combiner can be statistically secure.*

Proof: For Π_{MPC}, we rely on the protocol of [38], which is statistically t-secure if $t < n/2$ (and employs a broadcast channel). For Π_{2party}, we can use the protocol of [31], which provides statistical security in the malicious OT-hybrid model. ∎

4.1 Constant-Rate OT-Combiners in the Malicious Model

The OT complexity of the combiner in the malicious model is higher than in the semi-honest model. This is because of the inherent complexity of the underlying protocols Π_{MPC} and Π_{2party} and because of the employment of authentication, which requires secure two-party computation of functionalities involving MACs. The underlying principles that allow for constant-rate combiners in the malicious model are the following:

- Use a (constant-round) protocol Π_{MPC} in which the overall communication of the servers is $O(\ell)$. This, in turn, translates to the size of inputs that the

split-servers can run Π_{2party} on. Such a protocol for arbitrary NC^0 functionalities (including OT^ℓ as a special case) is described in [30], building on [16]. In contrast to the semi-honest model, where each server can receive just a constant number of field elements, here we need ℓ to be sufficiently larger than n, say $\ell = nk$ for a security parameter k. In such a case, each server will receive $O(k)$ field elements. This, in turn, will translate into a larger (non-constant) number of invocations of each candidate.
– Use Π_{2party} whose (amortized) OT complexity is a constant multiple of the input length I. Such a protocol was recently given in [33]. This protocol provides computational security and invokes $O(I+k)$ instances of *string-OT* with strings of length k (rather than bit-OT) along with a one-way function.

The above properties assure that the total number of calls to the (string-)OT-candidates is $O(\ell)$, provided that the MAC does not add a substantial overhead. The latter is guaranteed by the fact that the messages sent in the underlying protocol Π_{MPC} are long enough. Thus, the use of MACs does not increase the asymptotic length of the inputs.

A remaining caveat is that even though each of our candidates is a string-OT, the $O(\ell)$ calls to the candidates produce ℓ instances of *bit-OT*. Indeed, in the protocol Π_{split} obtained by our general compiler the length of the views of both S_i and R_i will be proportional to the total length of all strings (rather than the number of OTs). Thus, implementing ℓ good string-OTs would require $O(k\ell)$ calls to the candidates. To reduce the number of calls to $O(\ell)$, we observe that it is possible to modify our generic implementation of Π_{split} so that in all invocations of Π_{2party} the inputs of the R_i's are short, namely of total size $O(\ell)$ (assuming $\ell = kn$), whereas the inputs of the S_i's are of total size $O(k\ell)$. Since the number of string OTs required by [33] is determined only by the length of the receiver's input, we will end up using $O(\ell)$ calls to string-OT candidates to produce ℓ good string-OTs. Overall we get:

Theorem 8. *There is a computationally secure combiner for* string-OT *with constant production rate and constant tolerance ratio in the* malicious model. *The combiner makes $O(k)$ calls to each OT-candidate, as well as black-box use of a one-way function.*

5 Application: Constant-Rate OTs from a Noisy Channel

In this section, we apply our constant-rate combiners for the semi-honest model in order to efficiently produce a reliable stream of bit-OTs from a noisy channel, namely a binary symmetric channel which flips each bit with probability p. (We will refer to the latter channel as a BSC with crossover probability p.) Known constructions for this task [14, 13, 17] require sending $\Omega(k)$ bits over the channel in order to generate just a single OT call, even in the semi-honest model. We show that, using our constant-rate combiners, one can achieve a number of OT calls that is a constant multiple of the number of bits sent over the channel. Formally:

Theorem 9 (Constant-rate OTs from a noisy channel). *For any constant $0 < p < 1/2$, there exists a two-party protocol that securely implements the OT^ℓ functionality in the semi-honest model by having the parties communicate $O(\ell)$ bits over a BSC with crossover probability p. The protocol has perfect privacy and statistical correctness, where the error probability is $2^{-\Omega(\ell)}$.*

Proof: The idea is that, instead of using a constant number of noisy bits to produce a single secure OT, we produce an instance of OT that has perfect privacy but a small constant error probability. Each such OT can in turn be viewed as an *OT-candidate*. These candidates are then combined, using our constant-rate error-tolerant combiner, to give a linear number of good OT calls (this time with exponentially small error). We use the combiner from Theorem 6, that has constant rate and constant tolerance ratio and makes just $O(1)$ calls to each candidate. Note that we can view each call to a candidate as a distinct candidate, at the cost of further reducing the tolerance ratio to some small constant $\epsilon > 0$.

We now present a variant of a protocol from [14] that can implement OT with perfect privacy and an arbitrarily small constant error $\epsilon > 0$ by communicating a constant number of bits over the BSC. By known reductions, it suffices to implement such OT on random inputs. Let w, z be sufficiently large constants. The sender, S, picks z random bits $r_1, \ldots, r_z$ and sends each one $2w$ times to R (we call each corresponding sequence of $2w$ bits received by R a "block"). A block is of "type I" if it has an equal number of 0's and 1's and is of "type II" if it has only 0's or only 1's. Since w is a constant, we expect a constant fraction of blocks of each type (this fraction is a function of w and the noise level p) and hence by increasing z we can guarantee that both types exist with high probability. Now the receiver assigns a block of type I to the sender's bit it should not learn and a block of type II to the sender's bit it should learn. As required, this gives perfect privacy (namely, R does not learn any information on the bit it should not learn), but has a small ($2^{-\Omega(w)}$) probability of error in the bits R should learn. We finally note that all the additional (reliable) communication required by the combiner can be implemented via standard error-correcting codes by communicating $O(\ell)$ bits over the noisy channel. ∎

6 Application: Extending OTs Efficiently

In this section, we present *efficient* black-box reductions of $\mathrm{OT}^{p(k)}$ to $\mathrm{OT}^{q(k)}$ in the malicious model, where $q(k)$ is a *fixed* polynomial and $p(k)$ is *any* polynomial. (Throughout this section, OT refers to *string-OT* of k-bit strings.) The reductions make an additional black-box use of a cryptographic hash function and build on a protocol from [29] for the semi-honest model.

We give two distinct reductions, each making just a constant number of calls to the hash function per each OT call generated. The first follows by applying our combiners after running a cut-and-choose procedure over the protocol of [29]. As in [29], the protocol uses a so-called *correlation-robust hash function*

(CorRH): an explicit function h such that for random strings $s, t_1, \ldots, t_m$ the distribution $(h(s \oplus t_1), \ldots, h(s \oplus t_m), t_1, \ldots, t_m)$ is pseudorandom (see [29] for further discussion). Alternatively, a non-programmable, non-extractable random oracle suffices to instantiate the CorRH. This solution requires a seed of k^3 OTs (more precisely, $O(k\sigma^2)$ OTs, where σ is a statistical security parameter) rather than the k OTs required in the semi-honest model.

Theorem 10 (Informal). *Let k be a security parameter. For any polynomial $p(k)$, there exists a black-box reduction of $OT^{p(k)}$ to OT^{k^3} in the malicious model, under the CorRH assumption. The construction requires only a constant number of calls to the hash function per each OT produced.*

Proof sketch: Consider the following ideal functionality, denoted IKNP^ℓ (as it captures a core idea of [29]): the sender S has input $\mathbf{a} = (a_1, \ldots, a_k) \in \{0,1\}^k$. For each $j \in [\ell]$, the receiver R has inputs $\mathbf{b}^j = (b_1^j, \ldots, b_k^j) \in \{0,1\}^k$ and $\mathbf{m}^j = (m_1^j, \ldots, m_k^j) \in \{0,1\}^k$. For $j \in [\ell]$, the sender has output $\mathbf{d}^j = \mathbf{a} \wedge \mathbf{b}^j \oplus \mathbf{m}^j$, where $\wedge$ and $\oplus$ denote bitwise operations. We also consider a committed version, called CIKNP^ℓ, where parties are also committed to their inputs (if a party inputs a special symbol `reveal!`, its inputs will be leaked to the other party). Another stepping stone is a special version of OT^ℓ, called SOT^ℓ. It works as OT^ℓ, except that a *malicious* R may give a special input `cheat!` before inputs are provided by S. In response to this, R will receive *all* inputs of S. Later, R can give another special input `open!` in response to which S is told whether R at some point input `cheat!`. As a side effect, `open!` leaks the choice bits of R.

The proof follows a series of reductions. First, CIKNP^ℓ is constructed from OT^{k^2} (using just a single call to OT^{k^2}). This step follows the reduction from [29] of IKPN^ℓ to OT^k while the commitment property is achieved in the natural way by using k *committed OT*s [15] as the underlying primitive. As shown in [15], a committed OT can be implemented, in a black-box way, using $O(k)$ OTs and thus the overall $O(k^2)$ OTs. The second step, which is detailed below, builds SOT^ℓ from CIKNP^ℓ. Finally, one constructs $\mathrm{OT}^{p(k)}$ by making k calls to an instance of SOT^ℓ. The last step calls k instances of SOT^ℓ and, using a simple cut-and-choose technique, one can produce $O(k)$ instances of SOT^ℓ of which a sufficiently small constant fraction is insecure. Then, one applies our constant-rate combiner to get an implementation of $\mathrm{OT}^{O(k\ell)}$.

It remains to reduce SOT^ℓ to CIKNP^ℓ with an amortized constant number of hash function applications per produced OT. The protocol uses hash functions $H^j : \{0,1\}^k \to \{0,1\}^k$, for $j \in [\ell]$, where we let $H^j(x) = H(j \| x)$, for some fixed hash function H. Recall that SOT^ℓ takes as inputs secrets $(s_0^1, s_1^1), \ldots, (s_0^\ell, s_1^\ell)$ from the sender, and ℓ choice bits $c^1, \ldots, c^\ell$ from the receiver.

1. First, CIKNP^ℓ is called with S inputting a random $\mathbf{a}$ and R inputting a random $\mathbf{m}^j$ and $\mathbf{b}^j = (b^j, \ldots, b^j)$, where $b^j = c^j$ is the choice bit of R.
2. For $j \in [\ell]$, the sender S computes $\mathbf{d}_0^j \leftarrow \mathbf{d}^j$, $\mathbf{d}_1^j \leftarrow \mathbf{d}^j \oplus \mathbf{a}$, $r_0^j = H^j(\mathbf{d}_0^j)$ and $r_1^j = H^j(\mathbf{d}_1^j)$, and R computes $\mathbf{d}_{b^j}^j \leftarrow \mathbf{m}^j$ and $r_{b^j}^j = H^j(\mathbf{d}_{b^j}^j)$.
3. For $j \in [\ell]$, the sender S sends $e_0^j = r_0^j \oplus s_0^j$ and $e_1^j = r_1^j \oplus s_1^j$ to R, and R outputs $s_{b^j}^j = e_{b^j}^j \oplus r_{b^j}^j$.

To implement the `open!` command, the receiver will input `reveal!` to CIKNP$^\ell$ to show the values $\mathbf{m}^j$ and $\mathbf{b}^j$ to S. Sender S considers it a cheat if any $\mathbf{b}^j$ is not one of the monochromatic vectors 0^k or 1^k.

Correctness and security against a malicious sender are straightforward. The security against a malicious R is shown by a simulator with access to SOT$^\ell$. At a high level, if any of the $\mathbf{b}^j$ input by R is polychromatic, then the simulator inputs `cheat!` to SOT$^\ell$, learns the inputs of S and uses these to run the rest of the simulation as in the protocol. If, on the other hand, all $\mathbf{b}^j$ are monochromatic, then the simulator is reminiscent of that of [29] (including the use of the CorRH assumption). A complete proof appears in the full version. ∎

The second result manages to work with a seed of just k OTs (rather than the k^3 OTs used in Theorem 10). For this result, we use a natural generalization of the CorRH assumption, called the *generalized correlation-robust hash function (GCorRH) assumption* and a more specialized variant called the *special xor correlation-robust hash function (S⊕CorRH) assumption*. As with the CorRH assumption, it holds that a random function satisfies the new assumption with overwhelming probability. This, in particular, implies the security of our protocol in the non-programmable, non-extractable random oracle model. We stress, however, that all assumptions are concrete computational assumptions, and our proofs are in the standard model.

Theorem 11. *Let k be a computational security parameter. For any polynomial $p(k)$, there exists a black-box reduction of $OT^{p(k)}$ to OT^k in the malicious model under the GCorRH and S⊕CorRH assumptions. The construction requires only a constant number of calls to the hash function per each OT produced.*

At a high level, we notice from the proof of the previous theorem that in order to gain an advantage, a cheating R must pick some $\mathbf{b}^j$ to be polychromatic. Note that $\mathbf{d}_0^j = \mathbf{a} \wedge \mathbf{b}^j \oplus \mathbf{m}^j$ and $\mathbf{d}_1^j = \mathbf{a} \wedge \bar{\mathbf{b}}^j \oplus \mathbf{m}^j$, where $\bar{\mathbf{b}}^j = 1^k \oplus \mathbf{b}^j$. This means that, when $\mathbf{b}^j$ is polychromatic, both $\mathbf{d}_0^j$ and $\mathbf{d}_1^j$ depend on some bits of $\mathbf{a}$. The honest R will always know $\mathbf{d}_{b^j}^j = \mathbf{m}^j$. We exploit this difference by introducing a test where R, for each j, shows that it knows $\mathbf{d}_0^j$ or $\mathbf{d}_1^j$ without revealing which. Essentially, we let S send the first k bits of each $H^j(\mathbf{d}_0^j) \oplus H^j(\mathbf{d}_1^j)$ to R (suppose H^j has $2k$-bit outputs). R must then return the first k bits of $H^j(\mathbf{d}_0^j)$. This is easy for an honest R, which knows $H^j(\mathbf{d}_{b^j}^j)$, but will catch a cheating R with some probability related to how many bits of $\mathbf{a}$ the receiver needs to guess one of $\mathbf{d}_0^j$ and $\mathbf{d}_1^j$. This test, however, introduces an opening for S to cheat, which requires an extra fix. After these tests, we have (with overwhelming probability) at most k bad OTs out of the ℓ OTs being produced, and we remove the bad OTs by using our combiner. The details and proof appear in the full version.

References

1. D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th STOC*, pp. 479–488, 1996.

2. D. Beaver. Commodity-based cryptography. In *29th STOC*, pp. 446–455, 1997.
3. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th STOC*, pp. 1–10, 1988.
4. G. Brassard, C. Crépeau, and J.-M. Robert. All-or-nothing disclosure of secrets. In *Crypto '86, LNCS*, vol. 263, pp. 234-238, 1987.
5. G. Brassard, C. Crépeau, and S. Wolf. Oblivious Transfers and Privacy Amplification. *J. Cryptology* 16(4), pp. 219-237, 2003.
6. R. Canetti. Security and composition of multiparty cryptographic protocols. In *J. of Cryptology*, 13(1), 2000.
7. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th STOC*, pp. 11–19, 1988.
8. H. Chen and R. Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *Crypto '06, LNCS*, vol. 4117, pp. 521–536, 2006.
9. H. Chen, R. Cramer, S. Goldwasser, R. de Haan, and V. Vaikuntanathan. Secure computation from random error correcting codes. In *Eurocrypt '07, LNCS*, vol. 4515, pp. 291–310, 2007.
10. B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM Journal on Disc. Math.*, 4(1):36–47, 1991. preliminary version in STOC 89.
11. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *2nd Theory of Cryptography Conference – (TCC '05)*, pp. 342–362, 2005.
12. R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Eurocrypt'00, LNCS*, vol. 1807, pp. 316–334, 2000.
13. C. Crépeau. Efficient cryptographic protocols based on noisy channels. In *Eurocrypt '98*, pp. 306–317, 1997.
14. C. Crépeau and J. Kilian. Achieving oblivious transfer using weakened security assumptions. In *29th FOCS*, pp. 42–52, 1988.
15. C. Crépeau, J. van de Graaf, and A. Tapp. Committed oblivious transfer and private multi-party computation. In *Crypto '95, LNCS*, vol. 963, pp. 110–123, 1995.
16. I. Damgård and Y. Ishai. Scalable secure multiparty computation. In *Crypto '06, LNCS*, vol. 4117, pp. 501–520, 2006.
17. I. Damgaård, S. Fehr, K. Morozov, and L. Salvail. Unfair Noisy Channels and Oblivious Transfer. In *Proc. first TCC*, pp. 355-373, 2004.
18. I. Damgård, J. Kilian, and L. Salvail. On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In *Eurocrypt '99, LNCS*, vol. 1592, pp. 56–73, 1999.
19. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.
20. M. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *24th STOC*, pp. 699–710, 1992.
21. O. Goldreich. **Foundations of Cryptography** - *Volume 2.* Cambridge University Press, 2004.
22. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *19th STOC*, pp. 218–229, 1987.
23. O. Goldreich and R. Vainish. How to solve any protocol problem - an efficiency improvement. In *Crypto '87, LNCS*, vol. 293, pp. 73–86, 1987.

24. D. Harnik, Y. Ishai, and E. Kushilevitz. How many oblivious transfers are needed for secure multiparty computation? In *Crypto '07, LNCS*, vol. 4622, pp. 284–302, 2007.
25. D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen. On tolerant combiners for oblivious transfer and other primitives. In *Eurocrypt '05, LNCS*, vol. 3494, pp. 96–113, 2005.
26. A. Herzberg. On tolerant cryptographic constructions. In *CT-RSA*, pp. 172–190, 2005.
27. M. Hirt and U. Maurer. Robustness for free in unconditional multi-party computation. In *Crypto '01, LNCS*, vol. 2139, pp. 101–118, 2001.
28. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *21st STOC*, pp. 44–61, 1989.
29. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Crypto '03, LNCS*, vol. 2729, pp. 145–161, 2003.
30. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *39th STOC*, pp. 21–30, 2007.
31. J. Kilian. Founding cryptography on oblivious transfer. In *20th STOC*, pp. 20–31, 1988.
32. J. Kilian. A general completeness theorem for two-party games. In *23rd STOC*, pp. 553–560, 1991.
33. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Eurocrypt '07, LNCS*, vol. 4515, pp. 52–78, 2007.
34. R. Meier and B. Przydatek. On robust combiners for private information retrieval and other primitives. In *Crypto '06, LNCS*, vol. 4117, pp. 555–569, 2006.
35. R. Meier, B. Przydatek, and J. Wullschleger. Robuster combiners for oblivious transfer. In *Theory of Cryptography Conference – (TCC '07)*, vol. 4392 of *LNCS*, pp. 404–418, 2007.
36. B. Przydatek and J. Wullschleger. Error-tolerant combiners for oblivious primitives. Manuscript, Personal Communication, 2006.
37. M.O. Rabin. How to exchange secrets by oblivious transfer. TR-81, Harvard, 1981.
38. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st STOC*, pp. 73–85, 1989.
39. S. Wiesner. Conjugate coding. *SIGACT News*, 15(1):78–88, 1983.
40. J. Wullschleger. Oblivious transfer amplification. In *Eurocrypt '07, LNCS*, vol. 4515, pp. 555–572, 2007.
41. A. C. Yao. How to generate and exchange secrets. In *27th FOCS*, pp. 162–167, 1986.