Founding Cryptography on Oblivious Transfer – Efficiently

Yuval Ishai*[1], Manoj Prabhakaran**[2], and Amit Sahai***[3]

[1] Technion, Israel and University of California, Los Angeles. yuvali@cs.technion.il
[2] University of Illinois, Urbana-Champaign. mmp@cs.uiuc.edu
[3] University of California, Los Angeles. sahai@cs.ucla.edu

Abstract. Protocols for secure multiparty computation (MPC) can be divided into two types: (1) protocols that only guarantee security in the presence of an honest majority, and (2) protocols that guarantee security even when there is no honest majority. Known protocols of the first type are typically much more efficient than known protocols of the second type. This is true even in hybrid models that allow oracle access to ideal cryptographic primitives such as oblivious transfer (OT).

Motivated by this efficiency gap, we present an efficient compiler for transforming MPC protocols of the first type into MPC protocols of the second type in the OT-hybrid model. More generally, our compiler combines a protocol of the first type with a protocol of the second type, but where the security of the latter only needs to hold against *semi-honest* adversaries. This compiler follows the "MPC in the head" approach initiated by Ishai, Kushilevitz, Ostrovsky, and Sahai (STOC '07) and further developed by Harnik, Ishai, Kushilevitz, and Nielsen (TCC '08).

Combining our compiler with variants of protocols from the literature, we get several applications for secure two-party computation and for MPC with no honest majority. These include:

– *Constant-rate two-party computation in the OT-hybrid model.* Applying our compiler to a variant of an efficient MPC protocol of Damgård and Ishai (Crypto '06), We obtain a statistically (UC) secure two-party protocol in the OT-hybrid model that can evaluate a general circuit C of size s with a total communication complexity of $O(s)$. (For simplicity, we ignore from here on *additive* terms that depend polynomially on the security parameter k, the circuit depth, and $\log s$. These terms become dominated by the leading term in most typical cases of large circuits.) This improves over the $O(k^3 s)$ complexity of the best previous protocol, and matches the best asymptotic complexity in the semi-honest model. The above result generalizes to a constant number of parties.

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

** Partially supported by NSF grants CNS 07-16626 and CNS 07-47027.

*** This research was supported in part from NSF grants 0627781, 0716389, 0456717, and 0205594, a subgrant from SRI as part of the Army Cyber-TA program, an equipment grant from Intel, an Alfred P. Sloan Foundation Fellowship, and an Okawa Foundation Research Grant.

– *Extending OTs in the malicious model.* We obtain a protocol for generating many OTs from few OTs whose amortized cost in communication and cryptographic computation is a constant multiple of the efficient protocol for the semi-honest model given by Ishai, Kilian, Nissim, and Petrank (Crypto '03). This protocol can be used for efficiently implementing an off-line precomputation of OTs required by our protocols in the OT-hybrid model.
– *Black-box constructions for constant-round MPC with no honest majority.* We apply our general compiler to a variant of an MPC protocol of Damgård and Ishai (Crypto '05) to obtain general computationally secure MPC protocols in the OT-hybrid model that use only a constant number of rounds, and only make a *black-box* access to a pseudorandom generator. This provides a very different alternative to a similar result for the two party case that was recently obtained by Lindell and Pinkas (Eurocrypt '07), and gives the first constant-round protocols for three or more parties that only make a black-box use of cryptographic primitives (and avoid expensive zero-knowledge proofs).

1 Introduction

Secure multiparty computation (MPC) [35, 19, 4, 8] allows several mutually distrustful parties to perform a joint computation without compromising, to the greatest extent possible, the privacy of their inputs or the correctness of the outputs. MPC protocols can be roughly classified into two types: (1) ones that only guarantee security in the presence of an honest majority, and (2) ones that guarantee security[4] against an arbitrary number of corrupted parties.

A qualitatively important advantage of protocols of the second type is that they allow each party to trust nobody but itself. In particular, this is the only type of security that applies to the important case of secure two-party computation. Unfortunately, despite the practical appeal of such protocols, their efficiency significantly lags behind known protocols for the case of an honest majority. (For the potential efficiency of the latter, see the recent large-scale practical application of MPC in Denmark [5].) This is the case even when allowing parties to make free use of idealized cryptographic primitives such as bit commitment and oblivious transfer.

In this work we revisit the problem of founding secure two-party computation and MPC with no honest majority on oblivious transfer. Oblivious transfer (OT) [33, 16] is a two-party protocol that allows a receiver to obtain one out of two strings held by a sender, without revealing to the sender the identity of its selection. More precisely, OT is a secure implementation of the functionality which takes inputs s_0, s_1 from the sender and a choice bit b from the receiver, and outputs s_b to the receiver. Kilian [28] showed how to base general secure

[4] Concretely, in this type of protocols it is generally impossible to guarantee output delivery or even fairness, and one has to settle for allowing the adversary to abort the protocol after learning the output. However, in such a case the identity of a corrupted party is revealed to all honest parties.

two-party computation on OT. Specifically, Kilian's result shows that given the ability to call an ideal oracle that computes the OT functionality, Alice and Bob can securely compute an arbitrary function of their inputs with unconditional (statistical) security. We refer to secure computation in the presence of an ideal OT oracle as secure computation in the *OT-hybrid model*. Kilian's result was later generalized to the multi-party setting (see [12] and the references therein). Unfortunately, these constructions are quite inefficient and should mainly be viewed as feasibility results. Even protocols which achieve only computational security in the OT-hybrid model (e.g., [30]) still require communication complexity that grows with the product of circuit size and a polynomial in the security parameter.

When revisiting this problem we take a very different perspective from the one taken in the original works. Rather than being driven primarily by the goal of obtaining *unconditional security*, we are mainly motivated by the goal of achieving better *efficiency* for MPC in "the real world", when unconditional security is typically impossible or too expensive to achieve.[5]

Advantages of OT-based cryptography. There are several important advantages to basing cryptographic protocols on oblivious transfer, as opposed to concrete number-theoretic or algebraic assumptions.

- Preprocessing. OTs can be pre-computed in an off-line stage, before the actual inputs to the computation or even the function to be computed are known, and later very cheaply converted into actual OTs [1].
- Amortization. The cost of pre-computing OTs can be accelerated by using efficient methods for *extending OTs* [2, 24, 22]. In fact, the results of the current paper imply additional improvement to the asymptotic cost of extending OTs, and thus further strengthen this motivation.
- Security. OTs can be realized under a variety of computational assumptions, or even with unconditional security under physical assumptions. (See [32] for efficient realizations of UC-secure OT in the CRS model under various standard assumptions.) Furthermore, since the methods for extending OTs discussed above only require protocols to use a relatively small number of OTs, one could potentially afford to diversify assumptions by combining several candidate OT implementations [23].

1.1 Our Results

Motivated by the efficiency gap between the two types of MPC discussed above, we present an efficient general *compiler* that transforms MPC protocols with security in the presence of an honest majority into secure two-party protocols in the OT-hybrid model. More generally, our compiler uses an "outer" MPC protocol with security against a minority of *malicious* parties to add robustness

[5] Note, however, that our results still imply efficient unconditionally secure protocols under physical assumptions, such as off-line communication with a trusted dealer, secure hardware, or noisy channels.

to an “inner” two-party protocol (for a functionality defined by the outer protocol), whose security only needs to hold against *semi-honest* parties. The inner protocol can be in the OT-hybrid model. The final two-party protocol realizes the functionality of the outer protocol with security against malicious parties in the OT-hybrid model, while only making a black-box use of the inner protocol. The compiler naturally generalizes to yield MPC protocols with more than two players and security in the presence of an arbitrary number of corrupted parties.

For instance, one can use our compiler with the BGW protocol [4] (in the malicious model) as the outer protocol and with the simple version of the GMW protocol [19, 18] in the *semi-honest* OT-hybrid model to obtain a simple derivation of Kilian’s result [28], with the additional benefit of providing fully simulatable (statistical) UC-security [7].

Combining our general compiler with variants of protocols from the literature, we get the following applications for secure two-party computation and MPC with no honest majority.

Constant-rate two-party computation in the OT-hybrid model. Using a variant of an efficient MPC protocol of Damgård and Ishai [14] combined with secret sharing based on algebraic geometric codes due to Chen and Cramer [9], we obtain a statistically UC-secure two-party protocol *in the OT-hybrid model* that can evaluate a general circuit C of size s with a total communication complexity of $O(s)$. (For simplicity, we ignore from here on *additive* terms that depend polynomially on the security parameter k, the circuit depth, and $\log s$. These terms become dominated by the leading term in most typical cases of large circuits.) This improves over the $O(k^3 s)$ complexity of the best previous protocol of Crépeau et al. [12], and matches the best asymptotic complexity in the semi-honest model.

Using preprocessing to pre-compute OTs on random inputs, the protocol in the OT-hybrid model gives rise to a protocol of comparable efficiency in the real world. Following off-line interaction that results in each party storing a string of length $O(s)$, the parties can evaluate an arbitrary circuit of size s on their inputs using $O(s)$ bits of communication and no cryptographic computations. Note that the preprocessing stage can be carried out offline, before the actual inputs are available or even the circuit C is known. Furthermore, the cost of efficiently implementing the off-line stage can be significantly reduced by using techniques for amortizing the cost of OTs on which we improve.

Unlike two-party protocols that are based on Yao’s garbled circuit method [35], our technique can be combined with a homomorphic encryption scheme to yield (computationally secure) protocols with a similar communication overhead for *arithmetic circuits*. This extension is often important for applications. However, in contrast to protocols based on Yao’s techniques, the above protocols cannot be implemented in a constant number of rounds and require $O(d)$ rounds for a circuit of depth d. It seems that in most typical scenarios of large-scale secure computation, the overall efficiency benefits of our approach can significantly outweigh the disadvantage in its round-complexity. We leave a more refined finetuning of our technique and comparison with alternative approaches for further study.

The above results generalize to a constant number of parties.

Extending OTs in the malicious model. Somewhat unexpectedly, our techniques for obtaining efficient cryptographic protocols which *rely* on OT also yield better protocols for *realizing* the OTs consumed by the former protocols. This is done by using an outer protocol which efficiently realizes the functionality implementing many instances of OT. Using an efficient OT extension protocol for the *semi-honest model* from [24] as an inner protocol, we can upgrade the security of this OT protocol to the malicious model with only a constant communication and cryptographic overhead. This improves over a recent result from [22] that obtains similar efficiency in terms of the number of hash functions being invoked, but worse asymptotic communication complexity. Our OT extension protocol can be used for efficiently implementing the off-line precomputation of OTs required by our protocols in the OT-hybrid model.

Black-box constructions for constant-round MPC with no honest majority. We combine our general compiler with a variant of a constant-round MPC protocol of Damgård and Ishai [13] to obtain general *computationally* UC-secure MPC protocols in the OT-hybrid model that use only a constant number of rounds, and only make a *black-box* access to a pseudorandom generator. This provides a very different alternative to a similar result for the two party case that was recently obtained by Lindell and Pinkas [30], and gives the first constant-round protocols for three or more parties that only make a black-box use of cryptographic primitives (and avoid expensive zero-knowledge proofs).

Additional results. In Section 5 we describe two additional applications: a constant-rate black-box construction of OT in the malicious model from OT in the semi-honest model (relying on the recent black-box construction of [25, 21]), and a construction of asymptotically optimal OT combiners [23] (improving over [22]). In the full version we present a two-party protocol in the OT-hybrid model that uses only a *single* round of OTs and no additional interaction. The protocol only makes $n + o(n)$ OT calls, where n is the size of the input of the party which receives the output.

1.2 Techniques

Our main compiler was inspired by the "MPC in the head" paradigm introduced by Ishai, Kushilevitz, Ostrovsky, and Sahai [26] and further developed by Harnik, Ishai, Kushilevitz, and Nielsen [22]. These works introduced the idea of having parties "imagine" the roles of other parties taking part in an MPC (which should have honest majority), and using different types of cross-checking to ensure that an honest majority really is present in the imagined protocol.

The central novelty in our approach is a surprisingly simple and robust enforcement mechanism that we call the "watchlist" method (or more accurately, the *oblivious* watchlist method). In describing our approach, we will refer to the case of two-party computation involving two "clients" A and B. In our compiler, an "outer" MPC protocol requiring an honest majority of servers is combined with an "inner" two-party computation protocol with security against only *semi-honest* adversaries. This is done by having the outer MPC protocol jointly "imagined" by the two clients, with each server's computation to be jointly simulated by the two clients, using the inner semi-honest two-party protocol to compute the next-message-functions for the servers. The only method we use to prevent cheating is that both clients maintain a watchlist of some fraction of the servers, such that client A will have full knowledge of the internal state of all servers in A's watchlist, while client B has no idea which servers are on A's watchlist. Then client A simply checks that the watchlisted servers behave as they should in the imagined outer MPC protocol. If a dishonest client tries to cheat for too many servers, then he will be caught because of the watchlist with overwhelming probability. On the other hand, because the outer MPC protocol is robust against many bad servers, a dishonest client *must* attempt to cheat in the computation of many servers in order to be able to gain any unfair advantage in the execution of the protocol.

One may wonder why this approach would lead to greater efficiency over, for example, standard "cut-and-choose" methods prevalent in cryptography. To describe this, we will make an analogy to error-correcting codes. In standard cut-and-choose protocols, one typically prepares many copies of some object, and then the other party can ask for "explanations" of several of these objects. Here, the overhead is clear – one has to prepare many copies of an object that will only be used once, analogous to a repetition-based error-correcting code. Underlying our approach are the more sophisticated error-correcting codes that are essential to MPC protocols in the honest majority setting. In our case, while we have to sacrifice some working components (our servers) due to the watchlists, the others perform useful work that is not wasted, and this allows us to get more "bang for the buck", especially in settings where amortization is appropriate.

2 Preliminaries

Model. We use the Universal Composition (UC) framework [7], although our protocols can also be instantiated in the stand-alone setting using the composability framework of citeCanetto00, Goldreich04book. The parties in the protocols have access to (private, point-to-point) communication channels, as well as possibly one or more ideal functionalities. (In particular, our final protocols are in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, where $\mathcal{F}_{\mathsf{OT}}$ is the ideal functionality providing the 1-out-of-2 string oblivious transfer functionality.) We consider protocols for realizing both standard and reactive functionalities.

We will use two types of multi-party computation protocols as ingredients: (1) Multi-party computation protocols in the honest majority setting secure against adaptive corruptions that are UC-secure [7]. Such protocols will be used for standard or reactive functionalities, depending on our goals. (2) Multi-party computation protocols secure only against honest-but-curious (also known as passive or semi-honest) adversaries (see [18] for definitions and simple examples), that are private if all but one party is secure. Furthermore, if we seek security against adaptive adversaries for our protocols, we will also need to assume that such semi-honest protocols are adaptively secure (even if all parties are corrupted).

Oblivious Transfer. The basic oblivious transfer primitive we rely on is a $\binom{2}{1}$ string-OT. However, in our protocols we shall refer to $\binom{q}{1}$ string-OT, as well as the Rabin string-OT (which delivers an input string to the receiver with a fixed probability δ). There are efficient and unconditional UC secure reductions (with constant communication overhead) of these primitives to $\binom{2}{1}$ bit-OT (implicit in [10, 6, 15, 11]). We generically refer to all these variants of OT as being provided by the functionality $\mathcal{F}_{\mathsf{OT}}$.

For later reference, we point out how a Rabin-string-OT with rational erasure probability p/q (for positive integers $p < q$) can be securely realized using $\binom{q}{1}$ string-OT: the sender inputs q strings to the $\binom{q}{1}$ string-OT, of which a random subset of p are the message being transferred and the rest are arbitrary (say the zero string); the receiver picks up one of the q strings uniformly at random; then the sender reveals to the receiver which p-sized subset had the string being transferred; if the receiver picked a string not belonging to this set, it outputs erasure, and else outputs the string it received.[6]

Our model of OT is asynchronous: multiple OT's executed in the same round can be executed in an arbitrary, adversarially controlled order. We note that while synchronous OT is a conceptually simpler model, asynchronous OT can be used to realize synchronous OT in a simple way.

3 Protocol Compiler

3.1 The Outer protocol $\overline{\Pi}$

Recall that our protocol compiler takes an "outer protocol" $\overline{\Pi}$ and produces a compiled m-party protocol $\Psi^{\mathcal{F}_{\mathsf{OT}}}$. Here we describe the requisite nature of $\overline{\Pi}$.

$\overline{\Pi}$ is a protocol among $n + m$ parties (we will use $n = \Theta(m^2 k)$, k being the security parameter for $\overline{\Pi}$), with m parties $\overline{C}_i$ $(i = 1, \ldots, m)$ designated as the *clients*, and the other parties $\overline{P}_j$ $(i = 1, \ldots, n)$ designated as the *servers*.

– *Functionality:* $\overline{\Pi}$ is a protocol for some functionality $\mathcal{F}$ (which could be a secure function evaluation, or possibly a reactive functionality) among the m clients. The servers do not have any inputs or produce any outputs.

---

[6] Note that the sender can "cheat" by using arbitrary inputs to the $\binom{p}{q}$ string-OT and declaring an arbitrary set as the p-sized subset containing the message. But this simply corresponds to picking one of the messages in the declared p-sized subset (considered as a multi-set) uniformly at random, and using it as the input to the p/q-Rabin-string-OT.

– *Security:* $\overline{\Pi}$ securely realizes the functionality $\mathcal{F}$, against up to $m-1$ client corruptions and *adaptive* active corruption of upto t servers. We will require $t = \Omega(n)$. The security is statistical.

– *Protocol Structure:* The protocol $\overline{\Pi}$ proceeds in rounds where in each round each party sends messages to the other parties and updates its states by computing on its current state, and then also incorporates the messages it receives into its state. Each server $\overline{P}_j$ maintains a state $\overline{\Sigma}_j$. For the sake of an optimization in our applications, we will write $\overline{\Sigma}_j$ as $(\overline{\sigma}_j, \overline{\mu}_{1\leftrightarrow j}, \ldots, \overline{\mu}_{m\leftrightarrow j})$, were $\overline{\mu}_{i\leftrightarrow j}$ is just the collection of messages between $\overline{C}_i$ and $\overline{P}_j$. We will refer to $\overline{\mu}_{i\leftrightarrow j}$ as the "local" parts of the state and $\overline{\sigma}_j$ as the "non-local" part of the state. Note that client $\overline{C}_i$ is allowed to know the local state $\overline{\mu}_{i\leftrightarrow j}$ of each server $\overline{P}_j$.

The servers' program in $\overline{\Pi}$ is specified by a (possibly randomized) function $\overline{\pi}$ which takes as input a server's current state and incoming messages from clients and servers, and outputs an updated state as well as outgoing messages for the clients and other servers. That is,[7]

$$\overline{\pi}(\overline{\sigma}_j; \overline{\boldsymbol{\mu}}_j; \overline{\mathbf{w}}_{\cdot\rightarrow j}; \overline{\mathbf{u}}_{\cdot\rightarrow j}) \rightarrow (\overline{\sigma}'_j, \overline{\mathbf{m}}'_{j\rightarrow\cdot}, \overline{\mathbf{u}}_{j\rightarrow\cdot}).$$

where $\overline{\boldsymbol{\mu}}_j = (\overline{\mu}_{1\leftrightarrow j}, \ldots, \overline{\mu}_{m\leftrightarrow j})$ is the vector of local states, $\overline{\mathbf{w}}_{\cdot\rightarrow j} = (\overline{w}_{1\rightarrow j}, \ldots, \overline{w}_{m\rightarrow j})$ is messages received in the previous round by server $\overline{P}_j$ from the clients, and similarly $\overline{\mathbf{u}}_{\cdot\rightarrow j} = (\overline{u}_{1\rightarrow j}, \ldots, \overline{u}_{n\rightarrow j})$ is the set of messages $\overline{P}_j$ received from the other servers. The outputs $\overline{\mathbf{m}}'_{j\rightarrow\cdot} = (\overline{m}'_{j\rightarrow 1}, \ldots, \overline{m}'_{j\rightarrow m})$ and $\overline{\mathbf{u}}_{j\rightarrow\cdot} = (\overline{u}'_{j\rightarrow 1}, \ldots, \overline{u}'_{j\rightarrow n})$ stand for messages to be sent by $\overline{P}_j$ to the clients and to the servers respectively. The output $\overline{\sigma}'_j$ is the updated (non-local) state of the server $\overline{P}_j$. The local states are updated (by definition) as $\overline{\mu}'_{i\leftrightarrow j} := \overline{\mu}_{i\leftrightarrow j} \circ (\overline{w}_{i\rightarrow j}, \overline{m}'_{j\rightarrow i})$.

3.2 The Inner Functionality $\mathcal{G}$ and the Inner Protocol ρ

We define an m-party secure function evaluation functionality $\mathcal{G}_j$ which will be used to "implement" server $\overline{P}_j$ by the clients $\overline{C}_i$ $(i = 1, \ldots, m)$. $\mathcal{G}_j$ works as follows:

– From each client $\overline{C}_i$ get input $(S_i, M_i, \overline{\mu}_{i\leftrightarrow j}, \overline{w}_{i\rightarrow j})$, where S_i will be considered an additive share of the non-local state $\overline{\sigma}_j$ of the server $\overline{P}_j$, and M_i an additive share of $\overline{\mathbf{u}}_{\cdot\rightarrow j}$, all the messages received by $\overline{P}_j$ from the other servers in the previous round.[8]

– Compute $S_1 + \ldots + S_m$, and $M_1 + \ldots + M_m$ to reconstruct $\overline{\sigma}_j$ and $\overline{\mathbf{u}}_{\cdot\rightarrow j}$. Evaluate $\overline{\pi}$ (as given in the above displayed equation) to obtain $(\overline{\sigma}'_j, \overline{\mathbf{m}}'_{j\rightarrow\cdot}, \overline{\mathbf{u}}_{j\rightarrow\cdot})$.

– To $\overline{C}_i$ give output $(S'_i, \overline{m}'_{j\rightarrow i}, M'_i)$ where $(S'_1, \ldots, S'_m)$ form a random additive sharing of the updated state $\overline{\sigma}'_j$ and $(M'_1, \ldots, M'_m)$ form a random additive sharing of the messages to the servers $\overline{\mathbf{u}}_{j\rightarrow\cdot}$.

[7] For the sake of brevity we have omitted the round number and server number as explicit inputs to $\overline{\pi}$. We shall implicitly use the convention that these are part of each component in the input.

[8] The additive sharing can be over bit strings, or more generally over an appropriate group, especially if we work in an arithmetic computational model.

We will need a protocol ρ (in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model) to carry out this computation. But the security requirement on this protocol is quite mild: *ρ securely realizes $\mathcal{G}_j$ against passive corruption (i.e., honest-but-curious adversaries).*

In all our applications, we shall exploit an important optimization in an inner protocol to implement $\mathcal{G}_j$. Suppose an invocation of $\overline{\pi}$ (i.e., for some server $\overline{P}_j$ and some round number) depends only on the local state $\overline{\mu}_{i\leftrightarrow j}$ and possibly $\overline{w}_{i\to j}$, does not change the state $\overline{\sigma}_j$, and is deterministic. We call such a computation a *type I computation* (all other computations are called type II computations). A simple secure implementation of $\mathcal{G}_j$ for type I computations involves the client $\overline{C}_i$ computing $(\overline{\mathbf{m}}'_{j\to\cdot}, \overline{\mathbf{u}}_{j\to\cdot})$ itself, and sending each client $C_{i'}$ as output $(X_{i'}, \overline{m}'_{j\to i'}, M'_{i'})$ for each party, where $X_{i'}$ is a random sharing of 0 and $M'_{i'}$ is a random sharing of $\overline{\mathbf{u}}_{j\to\cdot}$. The client $C_{i'}$ sets $S'_{i'} := S_{i'} + X_{i'}$. (This last step of adding a share of 0 is in fact redundant in our compiled protocol; we include it only for the sake of modular exposition.)

Thus what the compiler needs to be given as the inner protocol is an implementation of $\mathcal{G}_j$ only for type II computations. Then it is the computational complexity of type II computations that will be reflected in the communication complexity of the compiled protocol.

3.3 The compiled protocol

At a high-level, the compiled protocol has the following structure.

1. *Watchlists initialization:* Using OT, the following infrastructure is set up first: each honest client randomly chooses a set of k servers to put on its watchlist (which only that client knows). For each client i and server $\overline{P}_j$ there is a "watchlist channel" W_{ij} such that any of the clients can send a message in W_{ij}, and client $\overline{C}_i$ will receive this message if and only if server $\overline{P}_j$ is on its watchlist. As we shall see, the implementation of this will allow a corrupt client to gain access (albeit partial) to the watchlist channels of more than k servers. Nevertheless, we note that the total number of servers for which the adversary will have access to the watchlist channel will be $O(km^2) < t/2$.
We shall also require another variant of watchlist channel (that can be set up on top of the above watchlist channel infrastructure): for each server $\overline{P}_j$ there is a "watchlist broadcast channel" $\mathbf{W}_j$ such that any client can send a message on $\mathbf{W}_j$ and *all the clients* who have server $\overline{P}_j$ on their watchlists will receive this message. (Note that when there are only two clients, this variant is no different from the previous one.)
If the adversary has access to the watchlist channel for server $\overline{P}_j$, then we allow the adversary to learn which other clients have access to their watchlist channels for server $\overline{P}_j$. Jumping ahead, we remark that in this case we will consider server $\overline{P}_j$ as corrupted. By the choice of parameters this will corrupt at most $t/2$ servers (except with negligible probability).

2. *Simulating the execution of $\overline{\Pi}$:* Each client $\overline{C}_i$ plays the role of $\overline{C}_i$ in $\overline{\Pi}$. In addition, the clients will themselves implement the servers in $\overline{\Pi}$ as follows. At the beginning of each round of $\overline{\Pi}$, the clients will hold a secret sharing of the

state of each server. Then they will use the inner protocol to execute the server's next-message and state-evolution functions and update the shared state.

The purpose of the watchlists is two-fold: firstly it is used to force (to some extent) that the clients do not change their inputs to the inner protocol *between* invocations; secondly it is used to force honest behavior *within* the inner protocol executions. The actual use of watchlists is quite simple:

(a) To enforce consistency between invocations of the inner protocol, each client $\overline{C}_i$ is required to report over the watchlist broadcast channel $\mathbf{W}_j$ every message that it provides as input to or receives as output from every invocation of the inner protocol for $\mathcal{G}_j$.

(b) To enforce honest behavior within the protocol execution, each client is required to report over watchlist channels W_{ij} (for all i) every message that it receives within the invocation of the inner protocol for $\mathcal{G}_j$. Further, for each invocation of the inner protocol j, the watchlist broadcast channel $\mathbf{W}_j$ is used to carry out a "coin-tossing into the well" to generate the coins for each client to be used in that protocol. (This coin-tossing step is not necessary when certain natural protocols with a slightly stronger security guarantee — like the basic GMW protocol in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model — are used. See Remark 1 below.)

Any honest client who has server $\overline{P}_j$ in its watchlist must check the reported values from all clients for consistency. Note that at the beginning of the execution of the inner protocol, all clients are already committed to their inputs and randomness during the protocol. Further, all honest clients honestly report the messages received from the other protocols. As such a client watching server $\overline{P}_j$ has an almost complete view of the protocol execution, and it knows ahead of time exactly what messages should be reported over the watchlist channels in an honest execution. This is sufficient to catch any deviation in the execution, if the protocol uses only communication channels. *However*, if the protocol involves the use of OT channels (or more generally, other ideal functionalities) then it creates room for an adversary to actively cheat and possibly gain an advantage over passive corruption. Then the adversary can change its inputs to the OT functionality without being detected (or arrange the probability of being detected to depend on the inputs of honest clients). To prevent this kind of cheating, we shall force that if the adversary changes its input to the OT functionality, then with at least a constant probability this will produce a different output for an honest client (if the adversary is the sender in the OT), or (if the adversary is the receiver in the OT) the adversary will end up reporting a different output over the watchlist. This is easily enforced by using a simple standard reduction of OT to OT with random inputs from both parties.

Remark 1. A protocol which is secure against passive corruptions is not necessarily secure when the adversary can maliciously choose the random tape for the corrupt players. This is the reason our compiler needs to use a coin-tossing in the well step to generate the coins for the inner protocols. However some natural protocols remain secure even if the adversary can choose the coins. This is the case for perfectly secure protocols like the basic GMW protocol (in the

$\mathcal{F}_{\mathsf{OT}}$-hybrid model). When using such an inner protocol, the compiler can simply omit the coin-tossing into the well step.

Setting up the Watchlist Channels and Broadcast Channels. First we describe how the watchlist channels described above are set up using OTs, and then how to obtain watchlist broadcast channels using them. The basic idea is for the clients to pick up sufficiently long one-time pads from each other using OT, and later send messages masked with a fresh part of these one-time pads.

For this we shall be using Rabin-string-OT (i.e., erasure channel with a fixed erasure probability, and adequately long binary strings being the alphabet).

The construction of the watchlist channels is as follows: First each client randomly chooses a set of k servers to put on its watchlist. Next, each pair of clients (i', i) engages in n instances of δ-Rabin-string-OTs where client $C_{i'}$ sends a random string r_j (of length ℓ) to $\overline{C}_i$. By choice of $\delta = \Omega(k/n)$, we ensure that except with negligible probability $\overline{C}_i$ obtains the string in more than k of the n instances. (By the union bound, this will hold true simultaneously for all pairs (i', i), except with negligible probability.) Now, client $\overline{C}_i$ specifies to client $C_{i'}$ a random permutation σ on $[n]$ conditioned on the following: if j is in the watchlist of $\overline{C}_i$ and $\sigma(j) = j'$, then $r_{j'}$ was received by $\overline{C}_i$. Now, to send a message on the watchlist channel W_{ij}, the client $C_{i'}$ will use (a fresh part of) $r_{\sigma(j)}$ to mask the message and send it to $\overline{C}_i$. Note that if j is in the watchlist of client $\overline{C}_i$, then this construction ensures that $\overline{C}_i$ can read all messages sent on W_{ij} by any client. If the strings r_j are ℓ bits long then at most ℓ bits can be sent to the watchlist channel constructed this way.

Finally, we consider obtaining watchlist broadcast channel $\mathbf{W}_j$ from watchlist channels W_{ij} set up as above. To send a message on $\mathbf{W}_j$ first a client sends the message on W_{ij} for every i. Then each client $\overline{C}_i$ on receiving a message on a watchlist channel W_{ij} sends it out on $W_{i'j}$ for every $i' \neq i$. (If $\overline{C}_i$ does not have access to W_{ij}, it sends a special message (of the same length) to indicate this.) Then it checks if all the messages it receives in this step over W_{ij} are the same as the message it received in the previous step, and if not aborts.

It can be verified that the above constructions indeed meet the specification of the watchlist infrastructure spelled out in the beginning of this section.

Theorem 1. *For any m-party functionality $\mathcal{F}$, suppose $\overline{\Pi}$ is a protocol as specified in Section 3.1, with $n = \Theta(m^2 k)$ and $t = \Theta(k)$, for a statistical security parameter k. Let $\mathcal{G}$ be the functionality defined in Section 3.2. Then, given a protocol ρ that securely realizes $\mathcal{G}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model against passive corruptions, the compiled protocol described above securely realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrod model against active corruptions.*

If both $\overline{\Pi}$ and ρ statistically/computationally secure against adaptive/static corruption, then the compiled protocol inherits the same kind of security.

Note that $\mathcal{F}$ could be a secure function evaluation functionality, or a reactive functionality like commitment.

3.4 Proof Sketch

The proof of security for our compiler follows from a conceptually very simple simulator. Full details will be given in the full version of this paper; here we sketch a high-level overview of how our simulator works. At a very high level, the simulator's job is very simple: Since it simulates the OT channels that the adversary uses in the protocol, the simulator will have full knowledge of everything that is sent over the watchlists, as well as in every invocation of OT used within the inner protocol. Thus, the simulator will know immediately if the adversary causes any of the imagined servers to behave dishonestly. It is easy to argue that if the adversary cheats with respect to any server that is on an honest party's watchlist, then it will be caught with constant probability (this is enforced in part by the reduction of OT to OT with random inputs). Since each honest party's watchlist is large, this shows that if the adversary causes too many servers to behave dishonestly, it will be caught by an honest party with overwhelming probability.

To make this formal, the simulator will invoke Sim_{outer}, the simulator for the outer MPC protocol. The simulator will be allowed to corrupt up to t servers when interacting with Sim_{outer}. When the simulator observes that the adversary is trying to cause dishonest behavior by some server, then it corrupts that server (thereby learning the state and history of that server, allowing the simulator to finish the interaction with the adversary and provide appropriate output to it). As argued above, if the adversary causes dishonest behavior in too many servers, it will get caught with overwhelming probability, and therefore our simulator will not need to exceed t corruptions. The only caveat here is if the adversary simultaneously tries to cause cheating in too many servers (e.g. all the servers at once). To deal with this situation, we ensure that the adversary is caught *before* it receives any output, and so we can simulate the interaction with the adversary before we have to corrupt the corresponding server in the outer protocol. This follows in a straightforward way from the way that the watchlists are used and the fact that OT's are only used with random inputs.

4 A Constant-Rate MPC Protocol

The protocol involves n servers and two or more clients, where only clients have inputs and outputs. Most of our applications will rely on an outer MPC protocols with the following features.

- **Security:** The protocol is statistically UC-secure against an active adversary that may adaptively corrupt at most t servers and any subset of the client, where t is some constant fraction of n. For most of our applications we only need the weaker notion of "security with abort", allowing the adversary to abort the computation possibly after learning the output, but the protocol can be implemented (along the lines of [14]) to provide full security with the same asymptotic complexity.

- **Communication and round complexity:** A boolean circuit C of size s and depth d (with bounded fan-in) can be evaluated with a total communication complexity of $O(s) + \mathrm{poly}(n, k, d, \log s)$ bits, where k is a statistical security parameter.[9] Assuming broadcast as an atomic primitive, the protocol requires $O(d)$ rounds.
- **Computational complexity:** We are mainly concerned about the efficiency of computations done by the servers. These computations consist of two types:
  1. Type 1: Computation on an input that is known to one of the clients. In the protocol produced by our compiler, the client knowing the input will perform these computations, and correctness will be enforced via the watchlists. However, in the MPC protocol it is essential that these computations be done by the servers, since the clients cannot be trusted.
  2. Type 2: Local computations that involve secret information (which should not be revealed to clients) and consist of evaluating multiple instances of a *finite* function on disjoint inputs. (By a "finite function" we refer to a function whose domain size is an absolute constant.) These computations, typically of the form $ab + z$ where a, b, z are elements of an underlying finite field, are used for computing the server's next secret state and its outgoing broadcast message given its current secret state and the last incoming message. The total number of finite functions evaluated by all servers throughout the protocol execution is $O(s)+\mathrm{poly}(n, d)$ times a polynomial in the number of clients.

An MPC protocol as above can be obtained by combining a version of an MPC protocol from [14] with Algebraic-Geometric secret sharing over fields of constant size [9].[10] This combination directly yields a protocol with the above properties for $\mathbf{NC}^0$ circuits, which was recently used in [26] to obtain constant-rate zero-knowledge proofs and in [22] to obtain constant-rate OT combiners. In the full version of the paper we present the (natural) extension of this protocol that can be applied to arbitrary depth-d circuits, at the cost of requiring $O(d)$ rounds.

5 Applications

In this section we describe the main applications of our general compiler. These are mostly obtained by applying the compiler to variants of efficient MPC protocols and two-party protocols from the literature.

[9] While we do not attempt here to optimize the additive term, we note that a careful implementation of the protocol seems to make this term small enough for practical purposes. In particular, the dependence of this term on d can be eliminated for almost every "natural" circuit C.

[10] Using Franklin and Yung's variant of Shamir's secret sharing scheme [34, 17], as originally done in [14], would result in logarithmic overhead to the communication complexity of the protocol, and a polylogarithmic overhead in the complexity of the applications.

5.1 Constant-Rate Secure Computation in the OT-Hybrid Model

Our first application is obtained by instantiating the general compiler with the following ingredients. The outer protocol is an interactive variant of an MPC protocol from [14], combined with secret sharing over constant-size fields using AG codes [9]. The protocol and its required efficiency features are described in Section 4. The inner protocol can be taken to be the "semi-honest GMW" protocol in the OT-hybrid model [19, 18].

Theorem 2. *Let C be a boolean circuit of size s, depth d and constant fan-in representing an m-party functionality f for some constant $m \geq 2$. Then there is a statistically UC-secure m-party protocol realizing f in the OT-hybrid model whose total communication complexity (including communication with the OT oracle) is $O(s) + \mathrm{poly}(k, d, \log s)$, where k is a statistical security parameter, and whose round complexity is $O(d)$. Security holds against an adaptive adversary corrupting an arbitrary number of parties.*

The OTs required by the above protocol can be generated during a preprocessing stage at no additional cost. The multiplicative term in the round complexity can be minimized by using a suitable randomized encoding of the inner functions of the outer protocol. The above theorem extends to the case of a non-constant number of parties m, in which case the communication complexity grows by a multiplicative factor of $\mathrm{poly}(m)$. The theorem applies also to *reactive* functionalities, by naturally extending the outer protocol to this case. Finally, it can be extended to the case of *arithmetic* circuits (at the cost of settling for computational security) by using an inner protocol based on homomorphic encryption. We defer further details to the full version.

5.2 Black-Box Constructions for Constant-Round MPC with no Honest Majority

Traditional MPC protocols for the case of no honest majority followed the so-called GMW paradigm [19, 18], converting protocols for the semi-honest model into protocols for the malicious model using zero-knowledge proofs. Since such proofs are typically expensive and in particular make a non-black-box use of the underlying cryptographic primitives, it is desirable to obtain alternative constructions that avoid the general GMW paradigm and only make a black-box use of standard cryptographic primitives.

The protocols of [28, 12] (as well as the more efficient constructions from Section 5.1) achieve this goal, but at the cost of round complexity that depends on the depth of the circuit. The question of obtaining constant-round protocols with the same features remained open.

In the case of MPC with honest majority, this problem was solved by Damgård and Ishai [13], providing a black-box alternative to a previous protocol of Beaver, Micali, and Rogaway [3] that made a non-black-box use of a pseudorandom generator. The case of two-party computation was recently resolved by Lindell and

Pinkas [30] (see also [31, 29]), who presented a constant-round two-party protocol that makes a black-box use of (parallel) OT as well as a statistically hiding commitment. The question of extending this result to three or more parties remained open, as the technique of [30] does not seem to easily extend to more than two parties. Partial progress in this direction was recently made in [20].

By applying our compiler to a variant of the MPC protocol from [13], we obtain the following theorem:

Theorem 3. *For any $m \geq 2$ there exists an m-party* constant-round *MPC protocol in the OT-hybrid model which makes a* black-box *use of a pseudorandom generator and achieves computational UC-security against an active adversary which may adaptively corrupt at most $m-1$ parties.*

Note that unlike the protocol of [30] our protocol is UC-secure and does not rely on statistically hiding commitments. On the down side, it requires a larger number of OTs which is comparable to the circuit size rather than the input size, though the latter cost may be amortized using efficient methods for extending OTs (see Section 5.3) and moved to a preprocessing phase. We defer further optimizations of the protocol to the full version.

Proof sketch: The protocol from [13] is a general constant-round protocol involving n servers and m clients. It is adaptively, computationally UC-secure against an adversary that may corrupt an arbitrary strict subset of the clients and a constant fraction of the servers. Furthermore, players in this protocol only make a black-box use of a PRG, or alternatively a one-time symmetric encryption scheme. If all the invocations of the encryption scheme were done by clients, the claimed result would follow by directly applying our compiler with this protocol as the outer protocol (since local computations performed by clients remain unmodified by the compiler). While the protocol from [13] inherently requires servers to perform encryptions, it can be easily modified to meet the form required by our compiler. This is done by making the servers only perform encryptions where both the key and the message to be encrypted are known to *one* of the clients. Using the watchlist approach, the protocol produced by the compiler will make the corresponding client perform the encryption instead of the server.

For simplicity, we describe this modification for the case of two clients, Alice and Bob. This easily generalizes to an arbitrary number of clients m. In any case where a server in the protocol of [13] needs to broadcast an encryption of the form $E_k(m)$, it will instead do the following. The server parses the key k as a pair of keys $k = (k_A, k_B)$ and additively secret-shares the message m as $m = m_A + m_B$. Now it sends k_A, m_A to Alice and k_B, m_B to Bob (this is a dummy operation that is only used to argue security). Finally, the server broadcasts $E_{k_A}(m_A)$ and $E_{k_B}(m_B)$. Note that each of these two computations is of Type 1, namely it is done on values already known to one of the clients. Moreover, it is easy to see that the above distributed encryption scheme is still semantically secure from the point of view of an adversary that corrupts just one of the clients. Thus, the simulation argument from [13] (that only relies on the semantic security of E) applies as is. $\square$

5.3 OT Extension in the Malicious Model

Beaver [2] suggested a technique for extending OTs using a one-way function. Specifically, by invoking k instances of OT one can implement a much larger number n of OTs by making use of an arbitrary one-way function. A disadvantage of Beaver's approach is that it makes a non-black-box use of the one-way function, which typically makes his protocol inefficient. A black-box approach for extending OTs was suggested by Ishai, Kilian, Nissim, and Petrank [24]. In the semi-honest model their protocol has the following features. Following an initial seed of k string OTs (where k is a computational security parameter), each additional string OT only requires to make a couple of invocations of a cryptographic hash function (that satisfies a certain property of "correlation robustness"[11] as well as a PRG. The amortized communication complexity of this protocol is optimal up to a constant factor, assuming that each of the sender's strings is (at least) of the size of the input to the hash function. To obtain a similar result for the malicious model, [24] employed a cut-and-choose approach which multiplies the complexity by a statistical security parameter. A partial improvement was recently given in [22], where the overhead in terms of the use of the hash function was reduced to a constant, but the overhead to the communication remained the same. This result was obtained via the use of efficient OT combiners [23]. We improve the (amortized) communication overhead to be constant as well. While our result could be obtained via an improvement to the construction of OT combiners in [22] (see Section 5.4), we sketch here a simple derivation of the result by applying our compiler to the protocol for the semi-honest model in [24]. In the full version we will show an alternative, and self-contained, approach for obtaining a similar result by applying our general secure two-party protocol to an appropriate $\mathbf{NC}^0$ functionality.

The efficient OT extension protocol is obtained as follows. The outer protocol will be the MPC protocol from Section 4 with two clients, called a sender and a receiver, and k servers. The protocol will be applied to the following multi-OT functionality. The sender's input is an n-tuple of pairs of k-bit strings, and the receiver's input is an n-tuple of choice bits. The receiver's output is the n-tuple of chosen k-bit strings. This outer protocol can be implemented so that each of the k servers performs just a single Type 2 computation, consisting of an $\mathbf{NC}^0$ function with one input of length $O(n)$ originating from the sender and another input of length $O(n/k)$ originating from the receiver. Using a decomposable randomized encoding [27], each of these inner computations can be securely implemented (in the semi-honest model) using $O(n/k)$ OTs on k-bit strings. However, instead of directly invoking the OT oracle for producing the required OTs, we use the OT extension protocol for the *semi-honest* model from [24]. The two-party protocol obtained in this way realizes the multi-OT functionality with computational UC-security, and only makes a black-box use of a correlation-robust hash function as well as a seed of k^2 OTs (which also includes the OTs

[11] The correlation robustness property defined in [24] is satisfied by a random function. Arguably, it is sufficiently natural to render practical hash functions insecure if they are demonstrated not to have this property.

for initializing the watchlists). Its constant communication overhead (for $n \gg k$) is inherited from the outer and inner components. We defer further optimizations to the full version.

Black-Box Constructions of OT. Note that the above construction (before plugging in the protocol from [24]) has the feature that the inner protocol can make a *black-box* use of any OT protocol for the *semi-honest* model. This implies the following black-box approach for converting "semi-honest OTs" into "malicious OTs". First, make $O(k)$ black-box invocations of an arbitrary malicious OT to generate the watchlists. (Here and in the following, we allow a free black-box use of a PRG to extend a single OT on short strings, or few bit OTs, into OT on a long strings.) Then, make $O(n)$ *black-box* calls to any OT protocol for the semi-honest model to generate n instances of OT in the malicious model. The above black-box approach applies both to the UC and to the standalone model. Together with the black-box constructions of OT of Ishai, Kushilevitz, Lindell, and Petrank [25] and Haitner [21], we get a black-box construction of malicious OT in the *standalone model* from semi-honest OT with a *constant* amortized OT production rate. The constant rate applies both to the cases of bit-OT and string-OT.

5.4 OT Combiners

An OT combiner [23] allows obtain a secure implementation of OT from n OT candidates, up to t of which may be faulty. The efficiency of OT combiners was recently studied by Harnik, Ishai, Kushilevitz, and Nielsen [22], who obtained a construction for the semi-honest model that tolerates $t = \Omega(n)$ bad candidates and has a constant production rate, namely produces m good instances of OT using a total of $O(m)$ calls to the candidates. They also present a similar variant for the malicious model, but this variant has two weaknesses. First, the OTs being produced are only computationally secure (even if the good OT candidates have unconditional security, say by using semi-trusted parties or physical assumptions). Second, the communication complexity of the combiner protocol has a multiplicative overhead that grows polynomially with a cryptographic security parameter. Our approach can be used to eliminate both of these weaknesses, obtaining unconditionally secure OT combiners in the malicious model that tolerate $t = \Omega(n)$ bad candidates and have a constant production rate and a constant communication overhead.

We achieve the above by applying the protocol of Theorem 2 such that each OT which is associated with server i (both during the actual protocol and during the watchlist initialization) is implemented by invoking the i-th OT candidate. Unlike Theorem 2, here we need to rely on the robustness of the outer protocol (rather than settle for the weaker notion of "security with abort"). Another modification to the protocol of Theorem 2 is that the protocol is not aborted as soon as the first inconsistency is detected, but rather only aborts when there are inconsistencies involving at least, say, $t/10$ servers. This is necessary to tolerate incorrect outputs provided by faulty OT candidates. Since the faulty candidates can be emulated by an adversary corrupting the corresponding servers, we can afford to tolerate a constant fraction faulty candidates.

References

1. D. Beaver. Precomputing oblivious transfer. In D. Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 1995.
2. D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proc. 28th STOC*, pages 479–488. ACM, 1996.
3. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM, 1990.
4. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th STOC*, pages 1–10. ACM, 1988.
5. P. Bogetoft, D. L. Christensen, I. Damgard, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. `http://eprint.iacr.org/`.
6. G. Brassard, C. Crépeau, and M. Santha. Oblivious transfers and intersecting codes. *IEEE Transactions on Information Theory*, 42(6):1769–1780, 1996.
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Electronic Colloquium on Computational Complexity (ECCC) TR01-016, 2001. Previous version "A unified framework for analyzing security of protocols" availabe at the ECCC archive TR01-016. Extended abstract in FOCS 2001.
8. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th STOC*, pages 11–19. ACM, 1988.
9. H. Chen and R. Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.
10. C. Crépeau. Equivalence between two flavours of oblivious transfers. In C. Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 350–354. Springer, 1987.
11. C. Crépeau and G. Savvides. Optimal reductions between oblivious transfers using interactive hashing. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 201–221. Springer, 2006.
12. C. Crépeau, J. van de Graaf, and A. Tapp. Committed oblivious transfer and private multi-party computation. In D. Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123. Springer, 1995.
13. I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2005.
14. I. Damgård and Y. Ishai. Scalable secure multiparty computation. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 501–520. Springer, 2006.
15. Y. Dodis and S. Micali. Parallel reducibility for information-theoretically secure computation. In M. Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 74–92. Springer, 2000.
16. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

17. M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710. ACM, 1992.
18. O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.
19. O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In ACM, editor, *Proc. 19th STOC*, pages 218–229. ACM, 1987. See [18, Chap. 7] for more details.
20. V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2008.
21. I. Haitner. Semi-honest to malicious oblivious transfer - the black-box way. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 412–426. Springer, 2008.
22. D. Harnik, Y. Ishai, E. Kushilevitz, and J. B. Nielsen. OT-combiners via secure computation. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 393–411. Springer, 2008.
23. D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen. On robust combiners for oblivious transfer and other primitives. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 96–113. Springer, 2005.
24. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.
25. Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. Black-box constructions for secure computation. In *STOC*, pages 99–108. ACM, 2006.
26. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30. ACM, 2007.
27. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In *STOC*, pages 433–442. ACM, 2008.
28. J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31. ACM, 1988.
29. M. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Proceedings of 27th Symposium on Information Theory in the Benelux*, volume 3958 of *Lecture Notes in Computer Science*, pages 283–290. Springer, 2006.
30. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007.
31. P. Mohassel and M. K. Franklin. Efficiency tradeoffs for malicious two-party computation. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2006.
32. C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In these proceedings; available from Cryptology ePrint Archive, Report 2007/348, 2008. `http://eprint.iacr.org/`.
33. M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.
34. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), Nov. 1979.
35. A. C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE, 1986.