

Founding Cryptography on Oblivious Transfer – Efficiently

Yuval Ishai*

Technion, Israel and University of California, Los Angeles
yuvali@cs.technion.il

Manoj Prabhakaran†

University of Illinois, Urbana-Champaign
mmp@cs.uiuc.edu

Amit Sahai‡

University of California, Los Angeles
sahai@cs.ucla.edu

October 5, 2010

Abstract

We present a simple and efficient compiler for transforming secure multi-party computation (MPC) protocols that enjoy security only with an honest majority into MPC protocols that guarantee security with no honest majority, in the oblivious-transfer (OT) hybrid model. Our technique works by combining a secure protocol in the honest majority setting with a protocol achieving only security against *semi-honest* parties in the setting of no honest majority.

Applying our compiler to 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.** We obtain a statistically UC-secure two-party protocol in the OT-hybrid model that can evaluate a general circuit C of size s and depth d with a total communication complexity of $O(s) + \text{poly}(k, d, \log s)$ and $O(d)$ rounds. The above result generalizes to a constant number of parties.
- **Extending OTs in the malicious model.** We obtain a computationally efficient protocol for generating many string OTs from few string OTs with only a *constant amortized communication overhead* compared to the total length of the string OTs.
- **Black-box constructions for constant-round MPC with no honest majority.** We 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 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) [47, 25, 5, 13] 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

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

†Supported in part by NSF grants CNS 07-16626 and CNS 07-47027.

‡Research 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.

types: (1) ones that only guarantee security in the presence of an honest majority, and (2) ones that guarantee security¹ 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 case of secure two-party computation. Unfortunately, despite the 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 practical application of MPC in Denmark [6].) This is the case even when allowing parties to use 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) [45, 22] 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 [36] showed how to base general secure two-party computation on OT. Specifically, Kilian’s result shows that given the ability to call an ideal oracle that computes OT, two parties can securely compute an arbitrary function of their inputs with unconditional 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 [18] and the references therein). Unfortunately, these constructions are quite inefficient and should mainly be viewed as feasibility results.

When revisiting the problem of basing cryptography on OT, 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.² **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 [2].
- **AMORTIZATION.** The cost of pre-computing OTs can be accelerated by using efficient methods for *extending OTs* [3, 32, 30]. 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 [42] 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

¹Concretely, 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.

² 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.

number of OTs, one could potentially afford to diversify assumptions by combining several candidate OT implementations [31].

1.1 Our Results

Motivated by the efficiency gap between the two types of MPC discussed above, we present a simple and 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 and precisely, our compiler uses the following two ingredients:

- An “outer” MPC protocol Π with security against a constant fraction of *malicious* parties. This protocol may use secure point-to-point and broadcast channels. It realizes a functionality f whose inputs are received from and whose outputs are given to two distinguished parties.
- An “inner” two-party protocol ρ for a (typically simple) functionality g^Π defined by the outer protocol, where the security of ρ only needs to hold against *semi-honest* parties. The protocol ρ can be in the *OT-hybrid model*.

The compiler yields a two-party protocol $\Phi_{\Pi,\rho}$ which realizes the functionality f of the outer protocol with security against malicious parties in the OT-hybrid model. If the outer protocol Π is UC-secure [10] (as is the case for most natural outer protocols) then so is $\Phi_{\Pi,\rho}$. It is important to note that $\Phi_{\Pi,\rho}$ only makes a *black-box* use of the outer protocol Π and the inner protocol ρ ,³ hence the term “compiler” is used here in a somewhat unusual way. This black-box flavor of our compiler should be contrasted with the traditional GMW compiler [25, 24] for transforming a protocol with security in the semi-honest model into a protocol with security in the malicious model. Indeed, the GMW compiler needs to apply (typically expensive) zero-knowledge proofs that depend on the code of the protocol to which it applies. Our compiler naturally generalizes to yield MPC protocols with more than two parties which are secure (in the OT-hybrid model) in the presence of an arbitrary number of malicious parties.

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

Revisiting the classics. As a historically interesting example, one can obtain a conceptually simple derivation of Kilian’s result [36] by using the BGW protocol [5] (or the CCD protocol [13]) as the outer protocol, and the simple version of the GMW protocol in the *semi-honest OT-hybrid model* [25, 26, 24] as the inner protocol. In fact, since the outer protocol is not required to provide optimal resilience, the BGW protocol can be significantly simplified. The resulting protocol has the additional benefits of providing full simulation-based (statistical) UC-security and an easy generalization to the case of more than two parties.

Constant-rate two-party computation in the OT-hybrid model. Using a variant of an efficient MPC protocol of Damgård and Ishai [20] combined with secret sharing based on algebraic geometric codes due to Chen and Cramer [14] as the outer protocol, 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

³Furthermore, the functionality g^Π realized by ρ is also defined in a black-box way using the next-message function of Π . This rules out the option of allowing the compiler access to the code of f by, say, incorporating it in the output of g^Π .

terms become dominated by the leading term in most typical cases of large circuits.) This improves over the $O(k^3s)$ complexity of the best previous protocol of Crépeau et al. [18], and matches the best asymptotic complexity in the semi-honest model.

By using preprocessing to pre-compute OTs on random inputs, the protocol in the OT-hybrid model gives rise to a (computationally secure) protocol of comparable efficiency in the plain model. 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. The above results extend to the case of more than two parties, with a multiplicative overhead that grows polynomially with the number of parties.

Unlike two-party protocols that are based on Yao’s garbled circuit method [47], 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 its higher round-complexity.

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 that efficiently realizes a functionality which implements many instances of OT. More concretely, 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 [32]. Using the protocol from [32] inside the inner protocol, we can upgrade the security of this OT extension protocol to the malicious model with only a constant communication and cryptographic overhead. This improves over a recent result from [30] 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 all the 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 [19] 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 [38], 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 for malicious parties from OT for semi-honest parties (building on a recent black-box feasibility result of [34, 29]), and a construction of asymptotically optimal OT combiners [31] (improving over [30]). In Appendix B we present a two-party protocol in the OT-hybrid model that uses only a *single* round of OTs and no additional interaction. (This applies to functionalities in which only one party receives an output.) In the final version we present a stream-lined protocol which 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 [35] and further developed by Harnik, Ishai, Kushilevitz, and Nielsen [30]. 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. Our approach is similar to the construction of OT combiners from [30] in that it uses an outer MPC protocol to add privacy and robustness to an inner two-party protocol which may potentially fail.⁴ A major difference, however, is that our approach provides security in the malicious model while only requiring the inner protocol to be secure in the *semi-honest* model.

The central novelty in our approach is a surprisingly simple and robust enforcement mechanism that we call the “watchlist” method (or more appropriately, the *oblivious* watchlist method). In describing our approach, we will refer for simplicity 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. Each server’s computation is 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, since 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. Our watchlist-based method for enforcing honest behavior should be contrasted with the non-black-box approach of the GMW compiler [25] that relies on zero-knowledge proofs.

It is instructive to contrast our approach with “cut-and-choose” methods from the literature. In standard cut-and-choose protocols, one party typically prepares many instances of some object, and then the other party asks for “explanations” of several of these objects. A central difficulty in such an approach is to prevent the compromised instances from leaking information about secrets, while combining the un-compromised instances in a useful way (see e.g. [38]). In contrast, our approach achieves these goals seamlessly via the privacy and robustness of the outer MPC protocol. To see how our approach leads to efficiency improvements as well, we will make an analogy to error-correcting codes. In traditional cut-and-choose, 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 can be used in MPC protocols in the honest majority setting. 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.

⁴This idea is also reminiscent of the player virtualization technique of Bracha [7] and the notion of concatenated codes from coding theory.

2 Preliminaries

Model. We use the Universal Composition (UC) framework [10] to formalize and analyze the security of our protocols. (Our protocols can also be analyzed in the stand-alone setting, using the composability framework of [9, 24], or in other UC-like frameworks, like that of [43].) The parties in the protocols have access to (private, point-to-point) communication channels, as well as possibly one or more ideal functionalities such as OT or broadcast. The UC model is “asynchronous.” The order of activation of parties can be modeled as controlled by the environment (see [44]), and the communication channels among parties are adversarially controlled (modeled as ideal functionalities which explicitly allow the adversary to control them [11]). We shall follow the convention in [11] that the communication between the parties and the functionalities themselves is ideal (private and instantaneous), so that any non-ideal behavior (for example, that the adversary can block outputs) must be explicitly modeled as part of the functionality.

Secure Function Evaluation (SFE) functionalities. For most part, we will be dealing with multi-party secure function evaluation functionalities, defined in terms of a function f on m inputs, with m outputs (where m is the number of parties). We consider functionalities which allow the adversary to block the output to the honest players after receiving its own output. For clarity, we describe the functionality in more detail below:

Input phase: From each party P_i , the functionality accepts input (**input**, x_i). On receiving each input it notifies the adversary. The functionality remains in this phase until valid inputs (i.e., in the domain of f) from all m parties have been received.

Output phase: On exiting the input phase, the functionality computes $(y_1, \dots, y_m) = f(x_1, \dots, x_m)$. For each i such that P_i is corrupt, send (**output**, y_i) to the adversary. (If the adversary adaptively corrupts a player P_i , send y_i at that point.) Then, for each i such that P_i is honest, on receiving instruction (**deliver**, i) from the adversary send (**output**, y_i) to P_i .

We point out that even if y_i is the empty string, the output message is delivered to the party P_i . This serves as a confirmation to the party that all parties (including the corrupt ones) have sent their inputs to the functionality, and that the functionality has reached the output phase.

A consequence of the asynchronous UC model is that if multiple functionality instances are being used by parties in a protocol, they are not co-ordinated together, except as allowed by the functionalities. Thus the case of SFE functionalities, the adversary can, for instance, choose an order in which it sends inputs to the functionalities, obtaining output from one functionality before sending inputs to the next; the outputs to honest parties from all the functionalities can still be delivered together.

We shall also consider functionalities which are more general than the SFE functionalities described above. In a *randomized* SFE functionality, the function f also takes as input a “random tape,” a string that will be chosen uniformly at random by the functionality. A *reactive* functionality consists of repeated invocations of an SFE functionality, but provides an additional input to the function f , namely a “state” which is initially the empty string and is updated during each output phase.

Oblivious Transfer. The basic oblivious transfer primitive we rely on is a $\binom{2}{1}$ string-OT, referred to as OT. More precisely, OT is a 2-party SFE functionality (as defined above), specified by the function $f : (\bigcup_n (\{0, 1\}^n)^2) \times \{0, 1\} \rightarrow \{\epsilon\} \times \{0, 1\}^*$ defined as follows: $f((s_0, s_1), b) = (\epsilon, s_b)$ (where ϵ stands for the empty string). Here the length of the strings n is part of the functionality specification.

In our constructions we shall also employ $\binom{q}{1}$ string-OT. There are efficient and unconditionally UC-secure reductions with constant communication overhead of these primitives to $\binom{2}{1}$ bit-OT (implicit in [16, 8, 21, 17]). Hence, one could also assume bit-OT as our basic primitive. When settling for computational security, OT on long strings can be efficiently reduced to a single instance of OT on short strings via the use of a pseudorandom generator.

Our watchlist initialization protocol will use Rabin string-OT, which is a randomized SFE functionality: the function f takes a single string s from the sender (and the empty string from the receiver), and with a fixed probability δ , outputs (ϵ, s) , and with probability $1 - \delta$ outputs (ϵ, ϵ) . 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 with constant communication overhead. 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.⁵

Finally, we shall also rely on parallel access to multiple instances of OT— i.e., on a functionality OT^t which first executes the input phase of t instances of OT and then executes the output phase for all of them. As remarked above, simply using t instances of OT directly does not securely realize OT^t . However, the following simple protocol does realize OT^t , using t (unsynchronized) instances of OT:

1. First, the t OT sessions are run on random inputs $((r_0^i, r_1^i); c^i)$ (for $1 \leq i \leq t$).
2. Then for all i , Receiver sends $d^i = b^i \oplus c^i$, where b^i is its input for the i -th session in OT^n .
3. Then for all i , Sender sends $(x_0^i = s_{d^i}^i \oplus r_0^i, x_1^i = s_{1-d^i}^i \oplus r_1^i)$, and Receiver recovers $s_{b^i}^i = x_{c^i}^i \oplus r_{c^i}^i$.

3 Protocol Compiler

In this section we describe how to build a protocol $\Phi_{\Pi, \rho}^{\text{OT}}$ that securely realizes a functionality \mathcal{F} against active corruptions, using two component protocols Π and ρ^{OT} of weaker security. Π is a protocol for \mathcal{F} itself, but uses several servers and depends on all but a constant fraction of them being honest. ρ^{OT} is a protocol for a functionality \mathcal{G} (which depends on Π), but is secure only against passive corruptions. Below we describe the requirements on Π , ρ^{OT} and the construction of $\Phi_{\Pi, \rho}^{\text{OT}}$.

⁵Note 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.

3.1 The Outer Protocol Π

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

- *Functionality:* Π is a protocol for some functionality \mathcal{F} (which could be deterministic or randomized, and possibly reactive) among the m clients. The servers do not have any inputs or produce any outputs.
- *Security:* Π UC-securely realizes the functionality \mathcal{F} , against *adaptive* corruption of up to t servers, and either static or adaptive corruption of any number of clients (see Section 3.4.1). We assume static client corruption by default. We will require $t = \Omega(n)$. The corruptions are active (i.e., the corrupt parties can behave arbitrarily) and the security could be statistical or computational.
- *Protocol Structure:* The protocol Π proceeds in rounds. In each round:
 - Each party (client or server) sends messages to the other parties (over secure point-to-point channels) and updates its state by computing on its current state. For clarity of exposition, we shall add the restriction that the servers do not directly communicate among themselves, but only with the clients.⁶
 - Then it reads the messages received in this round, and incorporates them to its state. We shall require that (for honest parties) these messages are not erased from the state while the state is updated in the next round.

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 $(\overline{\sigma}_j, \overline{\mu}_{1 \leftrightarrow j}, \dots, \overline{\mu}_{m \leftrightarrow j})$, where $\overline{\mu}_{i \leftrightarrow j}$ 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. Our optimization stems from the fact that the 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 Π is specified by a (possibly randomized) function $\overline{\pi}$ which takes as input a server’s current state and incoming messages from the clients, and outputs an updated state as well as outgoing messages for the clients. That is,⁷

$$\overline{\pi}(\overline{\sigma}_j; \overline{\boldsymbol{\mu}}_j; \overline{\mathbf{w}}_{\cdot \rightarrow j}) \rightarrow (\overline{\sigma}'_j, \overline{\mathbf{m}}'_{j \rightarrow \cdot}). \quad (1)$$

where $\overline{\boldsymbol{\mu}}_j = (\overline{\mu}_{1 \leftrightarrow j}, \dots, \overline{\mu}_{m \leftrightarrow j})$ is the vector of local states, and $\overline{\mathbf{w}}_{\cdot \rightarrow j} = (\overline{w}_{1 \rightarrow j}, \dots, \overline{w}_{m \rightarrow j})$ is messages received in this round by server \overline{P}_j from the clients. The output $\overline{\mathbf{m}}'_{j \rightarrow \cdot} = (\overline{m}'_{j \rightarrow 1}, \dots, \overline{m}'_{j \rightarrow m})$ stands for messages to be sent by \overline{P}_j to the clients. 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 $\overline{\mu}'_{i \leftrightarrow j} := \overline{\mu}_{i \leftrightarrow j} \circ (\overline{w}_{i \rightarrow j}, \overline{m}'_{j \rightarrow i})$.

⁶We shall remove these restrictions later, and also allow the parties in Π to use broadcast/multicast channels. See Section ??.

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

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

In the compiled protocol, as we shall describe shortly, each client C_i will play the role of \overline{C}_i in a session of the outer protocol Π . In addition the clients C_1, \dots, C_m will collectively “implement” each of the servers \overline{P}_j ($j = 1, \dots, n$). We define a (possibly randomized) m -party reactive functionality \mathcal{G} which stands for the program of the servers; the instance corresponding to the j -th server \overline{P}_j will be denoted by \mathcal{G}_j .

We will need a protocol ρ^{OT} (in the OT-hybrid model) to carry out this computation. But the security requirement on this protocol is quite mild: ρ^{OT} *securely realizes \mathcal{G}_j against passive corruption (i.e., honest-but-curious adversaries)*. The security could be statistical or computational. Also, the security could be against adaptive corruption or static corruption (see Section 3.4.1). For the sake of round efficiency, our main construction requires ρ^{OT} to be secure against adaptive (passive) corruption (without which the number of rounds in our final protocol will need to be increased by a factor of $O(k)$; see Section 3.4.3).

In all our applications, we keep the communication complexity low by exploiting an important optimization possible in the inner protocol ρ^{OT} . Suppose that in \mathcal{G}_j , an invocation of $\overline{\pi}$ (for some round number) (1) depends only on the local state $\overline{\mu}_{i \leftrightarrow j}$ and possibly $\overline{w}_{i \rightarrow j}$ for some i , (2) does not change the state $\overline{\sigma}_j$, and (3) is deterministic. We call such a computation a *type I computation* (all other computations are called type II computations). Since C_i must have the local state $\overline{\mu}_{i \leftrightarrow j}$ and the message $\overline{w}_{i \rightarrow j}$ available locally, it can by itself carry out the computation of $\overline{\pi}$ and send the resulting messages to the other clients (without violating security against passive corruption). Thus, one can arrange that the communication complexity of the inner protocol reflects the computational complexity of only type II computations in $\overline{\pi}$.

3.3 The Compiled Protocol

The compiled protocol $\Phi_{\Pi, \rho}^{\text{OT}}$ will securely realize the functionality \mathcal{F}_a against active corruptions (like Π), but the only participants in this protocol are the m clients who have inputs and outputs (denoted by C_i , $i = 1, \dots, m$). $\Phi_{\Pi, \rho}^{\text{OT}}$ has the following two phases.

1. *Watchlists initialization:* In the first phase, using OT, the following “watchlist” 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). Then, for each server \overline{P}_j we will set up 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.

As we shall see, our implementation will allow a corrupt client to gain access (albeit partial) to the watchlist broadcast channels of more than k servers. Nevertheless, by an appropriate choice of parameters, we shall ensure that the total number of servers for which the adversary will have access to the watchlist channel will be $O(km^2) < t/2$ (except with negligible probability). If a corrupt client gains such access to \mathbf{W}_j , then we allow the adversary to learn which other clients have access to \mathbf{W}_j .

Jumping ahead, we remark that when the adversary gains access to \mathbf{W}_j , we will consider server \overline{P}_j as corrupted; this phase allows the adversary to corrupt less than $t/2$ servers in this way. We shall describe the implementation of this infrastructure in Section 3.3.1.

2. *Simulating the execution of Π :* In the second phase the clients start simulating a session of Π . Each client C_i plays the role of \overline{C}_i in Π . In addition, the clients will themselves implement

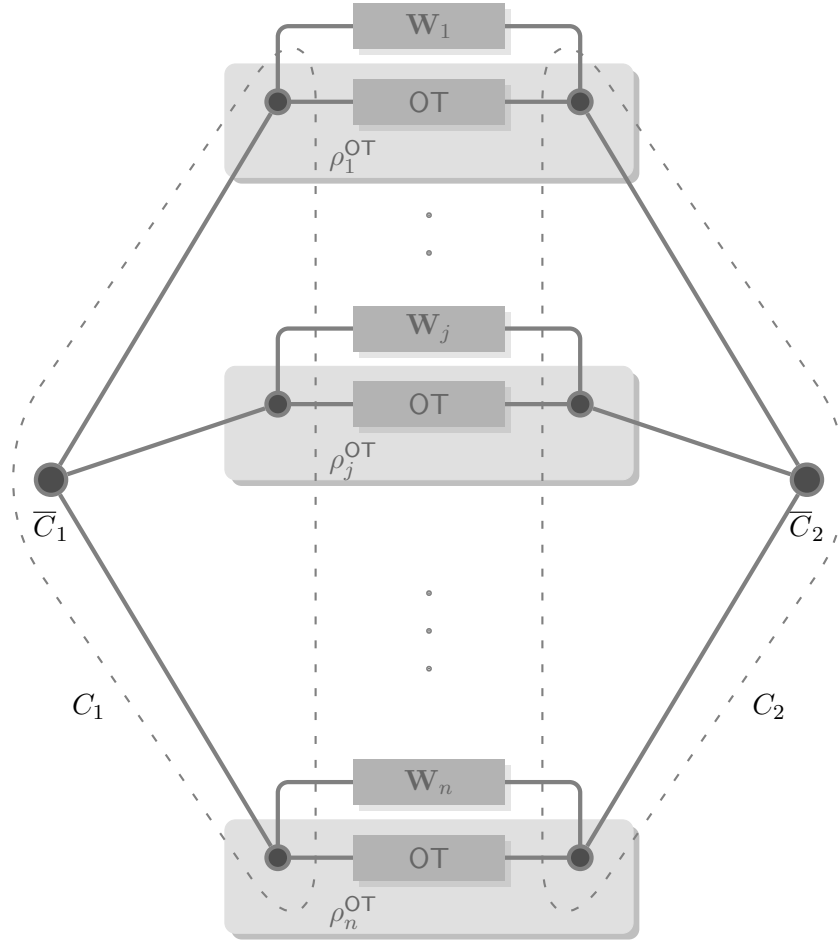


Figure 1: A schematic representation of the compiled protocol $\Phi_{\Pi, \rho}^{\text{OT}}$ (for a 2-party functionality). \bar{C}_1 and \bar{C}_2 are the clients in the outer protocol. Boxes labeled ρ_j^{OT} represent sessions of the inner protocol, involving two parties interacting via an OT channel and the watchlist channel \mathbf{W}_j . The programs enclosed by the dotted lines indicate the clients in the compiled protocol, C_1 and C_2 .

the servers in Π using the inner protocol ρ^{OT} for \mathcal{G} . We shall denote by ρ_j^{OT} the j -th session of ρ^{OT} , implementing \mathcal{G}_j , corresponding to the server \bar{P}_j in Π . At the beginning of each round, client C_i provides the message from \bar{C}_i to \bar{P}_j at that round as its input to ρ_j^{OT} .

The watchlists are used to force the clients (to some extent) to behave honestly in each instance of the inner protocol. In more detail:

- (a) For each invocation ρ_j^{OT} of the inner protocol, 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.⁸ That is, to generate a random tape for client C_i for use in

⁸This coin-tossing step is not necessary when certain natural protocols with a slightly stronger security guarantee — like the basic “passive-secure” GMW protocol in the OT-hybrid model — are used. See Remark 1 below.

ρ_j^{OT} , first all the clients (including C_i) send a share of the random tape over \mathbf{W}_j . Then all clients except C_i broadcast these shares (not over the watchlist channels). C_i uses the sum (say XOR) of all these shares (including its own) as the random tape in ρ_j^{OT} .

- (b) Each client is required to report over \mathbf{W}_j its inputs to the inner protocol session ρ_j^{OT} , as well as every protocol message that it receives within that session. This includes the messages received from the OT channels used within the protocol.

All honest clients are required to carry out consistency checks on the messages it can read from the various watchlists and the messages it receives in the rest of the protocol execution. If any of the following consistency checks fail, then the client is required to abort the execution of the entire (compiled) protocol. For concreteness, we shall require that after each round, the clients ensure that no one has aborted (using a round of acknowledgment message) before continuing.

Consistency checks: An honest client C_i who has server \bar{P}_j in its watchlist must do the following consistency checks between the messages reported over \mathbf{W}_j by all the clients and the messages in ρ_j^{OT} that it receives.

- First, C_i must ensure that the shares of the random tapes declared by all clients match the ones they have sent over \mathbf{W}_j . Then, it can compute the random tape for each client using the shares sent to \mathbf{W}_j .
- Using the random tape above and the inputs to ρ_j^{OT} reported on \mathbf{W}_j , C_i can compute all messages to be sent on ρ_j^{OT} by all the clients. C_i is required to check that all the messages it has access to are consistent with the calculated messages: this includes the messages it directly receives in ρ_j^{OT} (as messages on communication channels or as outputs from the OT functionality), and the messages received by all the honest clients in ρ_j^{OT} as reported by them over \mathbf{W}_j .

Note that a client watching server \bar{P}_j knows ahead of time exactly what messages should be received by each client in ρ_j^{OT} if all clients are honest. Also it sees the messages received by the clients (as reported by them). 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. In particular, the adversary can change its inputs to the OT functionality, deviating from what it announces on the watchlist channels, and arrange for 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 from OT over the watchlist than what it would have reported if it were honest. This is easily enforced by using a simple standard reduction of OT to OT with random inputs from both parties, as follows:

- All uses of OT in the inner protocol ρ^{OT} are replaced by the following subprotocol. First the sender uses a pair of random strings (r_0, r_1) as its input to OT and the receiver uses a random bit c . Then, the receiver sends $z := b \oplus c$ to the sender, where b is its original

input to the OT. The sender responds with $(r_0 \oplus x_z, r_1 \oplus x_{1 \oplus z})$. The receiver recovers $x_b := r_c \oplus (r_c \oplus x_{z \oplus c})$.

It is to such a modified inner protocol ρ^{OT} that we add the use of watchlist channels and checks.

Remark 1 (On tossing coins.) *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, typically, natural protocols with unconditional security remain secure even if the adversary can choose the coins. This is the case for perfectly secure protocols like the basic “passive-secure” GMW protocol (in the OT-hybrid model). When using such an inner protocol, the compiler can simply omit the coin-tossing into the well step.*

3.3.1 Setting up the Watchlist Broadcast Channels

First, using OT channels, we will implement simpler watchlist channels W_{ij} , for each client-server pair (C_i, \bar{P}_j) , such that any of the clients can send a message in W_{ij} , and C_i will receive this message if and only if server \bar{P}_j is on its watchlist. Then we shall use these channels to implement the watchlist broadcast channels \mathbf{W}_j . (Note that when there are only two clients, the two variants are equivalent.)

The basic idea in implementing W_{ij} 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). See Section 2 for implementation details.

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 C_i . By choice of $\delta = \Omega(k/n)$, we ensure that except with negligible probability 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 C_i specifies to client $C_{i'}$ a random permutation σ on $[n]$ conditioned on the following: if j is in the watchlist of C_i and $\sigma(j) = j'$, then $r_{j'}$ was received by 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 C_i . Note that if j is in the watchlist of client C_i , then this construction ensures that 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.

Now, we consider obtaining watchlist broadcast channel \mathbf{W}_j from watchlist channels W_{ij} set up as above. This is similar to how broadcast is obtained from point-to-point channels in [27]. To send a message on \mathbf{W}_j first a client sends the message on W_{ij} for every i . Then each client 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 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 $W_{i'j}$ are the same as the message it received in the previous step, and if not aborts.

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

Theorem 1 *Let \mathcal{F} be a (possibly reactive) m -party functionality. Suppose Π is an outer MPC protocol realizing \mathcal{F} , as specified in Section 3.1, with $n = \Theta(m^2k)$ and $t = \Theta(k)$, for a statistical security parameter k . Let \mathcal{G} be the functionality defined in Section 3.2 and ρ^{OT} a protocol that securely realizes \mathcal{G} in the OT-hybrid model against two-step passive corruption. Then the compiled protocol $\Phi_{\Pi,\rho}^{\text{OT}}$ described above securely realizes \mathcal{F} in the OT-hybrid model against active (static) corruptions. If both Π and ρ^{OT} are statistically/computationally secure, then the compiled protocol inherits the same kind of security.*

$\Phi_{\Pi,\rho}^{\text{OT}}$ has communication complexity $\text{poly}(m) \cdot (C_{\Pi} + nr_{\Pi}C_{\rho})$, round complexity $O(r_{\Pi}r_{\rho})$, and invokes OT $\text{poly}(m) \cdot nr_{\Pi}q_{\rho}$ times, where C_{Π} is the communication complexity of Π , r_{Π} is the number of rounds of Π , C_{ρ} is the communication plus randomness complexity of ρ^{OT} , r_{ρ} is the round complexity of ρ^{OT} , and q_{ρ} is the number of invocations of OT in ρ^{OT} .

Here by communication complexity of a protocol in the OT-hybrid model we include the communication with the OT functionality. By randomness complexity of a protocol we mean the total number of random bits used by (honest) parties executing the protocol. We remark that the complexity bounds given above can typically be tightened when analyzing specific inner and outer protocols.

PROOF: The proof of security for our compiler follows from a conceptually very simple simulator \mathcal{T} , which shows that the compiled protocol is as secure as an execution of the outer protocol in which no more than t servers are corrupted. \mathcal{T} plays the adversary in an outer protocol execution, and simulates an execution of the compiled protocol to the adversary \mathcal{A} . (See Figure 2.)

At a very high level, \mathcal{T} 's job is simple. Since \mathcal{T} simulates the OT channels that the adversary \mathcal{A} uses in the (simulated) execution of $\Phi_{\Pi,\rho}^{\text{OT}}$, it 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, \mathcal{T} will know immediately if and when the adversary deviates from honest behavior in the emulation of any of the servers. At that point, \mathcal{T} would (adaptively) corrupt that server in the outer protocol execution so that it can continue the simulation faithfully. Recall that we need to ensure that \mathcal{T} corrupts no more than t servers in the outer protocol execution. 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). Then, since each honest party's watchlist is large, if the adversary causes too many servers to behave dishonestly, it will be caught by an honest party with overwhelming probability, and will result in the protocol being aborted. Therefore the simulation also would have simulated an abort before having to corrupt too many servers.

Now we describe the simulation in more detail. We will consider an arbitrary environment in two scenarios. In the first scenario, the clients C_i ($i = 1, \dots, m$) take part in a session of the compiled protocol $\Phi_{\Pi,\rho}^{\text{OT}}$, with an adversary \mathcal{A} . In the other scenario the clients \bar{C}_i ($i = 1, \dots, m$) take part in a session of the outer protocol Π , along with n servers \bar{P}_j ($j = 1, \dots, n$), and \mathcal{T} which plays the role of the adversary in this session.

\mathcal{T} has the following structure. It internally runs a $\Phi_{\Pi,\rho}^{\text{OT}}$ session for \mathcal{A} (shown to the left in Figure 2) and lets \mathcal{A} interact with the environment directly. \mathcal{T} will simulate to \mathcal{A} the communication from all the honest clients, and the OT channels in $\Phi_{\Pi,\rho}^{\text{OT}}$. Externally, \mathcal{T} will run in a Π session (shown to the right in Figure 2), and will interact with the honest clients and the servers in that session. The simulation proceeds as follows.

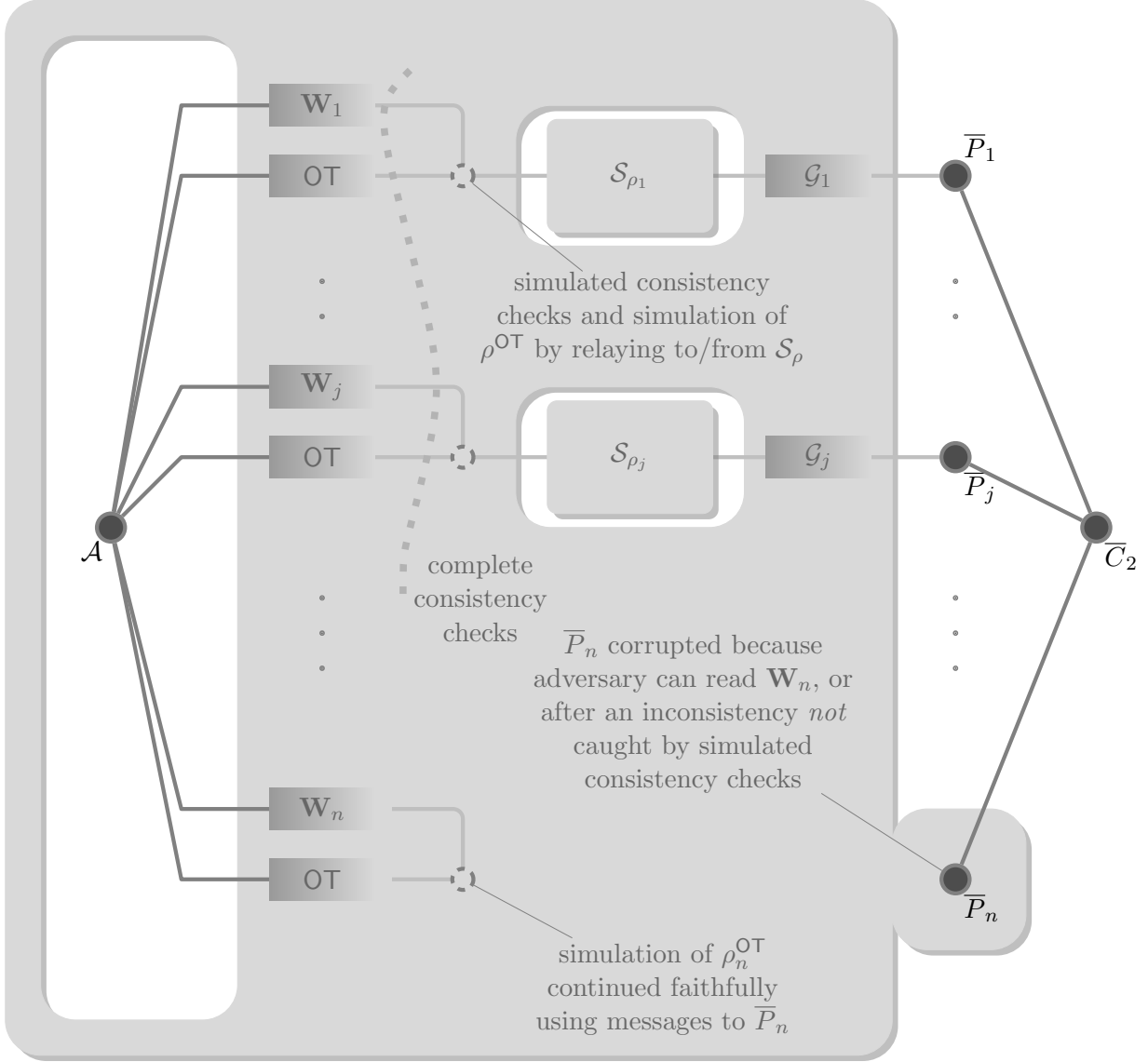


Figure 2: An outline of the simulator \mathcal{T} . Externally \mathcal{T} takes part in a session of the outer protocol (on the right). Internally, \mathcal{T} interacts with an adversary \mathcal{A} (which in turn interacts with the environment) in a simulated session of the compiled protocol (shown towards the left; cf. Figure 1). In each inner protocol session that is part of the simulated session of the compiled protocol, \mathcal{T} carries out consistency checks on everything that \mathcal{A} reports on the (simulated) watchlist channels. As long as an inner protocol session ρ_j^{OT} remains correct, \mathcal{T} uses the simulator \mathcal{S}_{ρ_j} (for honest-but-curious security of the inner protocol) to translate an interaction with the outer server \bar{P}_j into the interaction in ρ_j^{OT} . If an inconsistency it detects does not lead to aborting in ρ_j^{OT} , then \mathcal{T} corrupts the server \bar{P}_j on the right hand side to continue the simulation.

1. First, \mathcal{T} perfectly simulates the watchlist setup phase of $\Phi_{\Pi, \rho}^{\text{OT}}$, by playing the role of each uncorrupted client, interacting with the corrupted clients. \mathcal{T} simulates the OT channels used

for this.

While setting up the watchlist channels (see Section 3.3.1) each corrupt client C_i can, for each uncorrupt client $C_{i'}$, select $O(k)$ watchlists W_{ij} (except with negligible probability), such that it can read what $C_{i'}$ writes to W_{ij} . \mathcal{T} notes all such j , over all pairs of clients $(C_i, C_{i'})$ (corrupt and uncorrupt, respectively). If there are more than $t/2 = \Theta(m^2k)$ such values j , then \mathcal{T} fails and bails out; by the choice of the parameter $\delta = O(k/n)$, this occurs only with negligible probability. Otherwise (i.e., if there are at most $t/2$ such j), \mathcal{T} corrupts all those servers \bar{P}_j in the execution of Π .

Note that \mathcal{T} has access to all the watchlist broadcast channels (whereas each client in $\Phi_{\Pi,\rho}^{\text{OT}}$ has access to only the watchlist broadcast channels for the servers on its watchlist). Also, \mathcal{T} knows which all watchlist broadcast channels the adversary has (partial) access to. Further, \mathcal{T} has access to all inputs to the OT channels (where as clients have access to only the outputs they receive from the OT channels). It will rely on this extra information in the subsequent steps.

2. During the rest of the execution of the protocol, \mathcal{T} must simulate various kinds of messages in the compiled protocol $\Phi_{\Pi,\rho}^{\text{OT}}$. These messages fall into the following kinds:
 - (a) Client-to-client communication in the Π (outer protocol) session.
 - (b) Communication in the ρ^{OT} (inner protocol) sessions (which is used to emulate the servers in the outer protocol). Some of this communication is via OT channels.
 - (c) Communication in the watchlist channels.
 - (d) Abort messages.

To simulate the client-to-client communication in the outer protocol, \mathcal{T} passes on the messages between corrupted clients in the $\Phi_{\Pi,\rho}^{\text{OT}}$ session and the uncorrupted clients in the Π session unaltered. In order to simulate the messages sent to a watchlist broadcast channel \mathbf{W}_j to which the adversary has (partial) read access, \mathcal{T} will corrupt the server \bar{P}_j in the outer protocol right at the beginning, and with access to all the inputs of the honest clients in these sessions, will faithfully carry out the execution in the sessions. Simulating the communication in the remaining sessions of the inner protocol requires more care, and is described in detail below. At a high level, for each such session ρ_j^{OT} , *for as long as the adversary remains honest (but possibly curious) in ρ_j^{OT}* , \mathcal{T} will internally run a copy \mathcal{S}_{ρ_j} of the simulator for the inner protocol. Inputs for \mathcal{S}_{ρ_j} are derived from \bar{P}_j in the external outer protocol, and the messages that the adversary sends on the simulated watchlist channel \mathbf{W}_j . In addition, \mathcal{T} must be able to detect when the adversary deviates from honest behavior and be able to continue the simulation by simulating an abort, or simulating the honest clients continuing the protocol without detecting the deviation (for which it will corrupt the server \bar{P}_j). All this is carried out as follows:

- *Full consistency checks:* \mathcal{T} carries out the same consistency checks as the honest clients in $\Phi_{\Pi,\rho}^{\text{OT}}$ (i.e., consistency between messages in ρ_j^{OT} and \mathbf{W}_j for each j), but unlike an honest client, \mathcal{T} can carry out a “full” consistency check — i.e., detect any deviation from honest behavior in any inner protocol session — as it has access to all the inputs to the watchlist broadcast channels and to the inputs to the OT channels.

- *Input to and output of ρ_j^{OT}* : The simulator \mathcal{S}_{ρ_j} expects to interact with the functionality \mathcal{G}_j , which \mathcal{T} simulates by simply relaying messages to and from the external server \overline{P}_j . But further, being a simulator for passive adversaries, expects to be provided with the inputs to corrupt clients (which it will faithfully forward to the functionality \mathcal{G}_j). The input in the case of a corrupt client \overline{C}_i is a message $\overline{w}_{i \rightarrow j}$, from \overline{C}_i to \overline{P}_j . But C_i is required to send this input on the watchlist broadcast channel \mathbf{W}_j prior to using it in ρ_j^{OT} . If the consistency checks (as described above) pass, \mathcal{T} takes this input given to the watchlist broadcast channels as the corrupt clients' inputs. \mathcal{S}_{ρ_j} will then pass on these inputs to the simulated functionality \mathcal{G}_j , which are relayed to \overline{P}_j . In response, \mathcal{T} obtains the message $\overline{m}'_{j \rightarrow i}$ from \overline{P}_j , which it passes on to \mathcal{S}_{ρ_j} as the response from \mathcal{G}_j .
- *Execution of ρ_j^{OT}* : On receiving a message from a corrupt client in the inner protocol session ρ_j^{OT} , (message to an uncorrupted client or as input to the OT channel), if all the consistency checks (as described above) are passed, then \mathcal{T} passes on the ρ^{OT} message from the corrupt client to the inner simulator \mathcal{S}_{ρ_j} . Messages from \mathcal{S}_{ρ_j} to corrupt clients are passed on directly, as messages from the simulated ρ_j^{OT} session (i.e., as message from an uncorrupted client or as output from the OT channel).
- *Handling consistency check failures*: If some consistency check fails, \mathcal{T} faithfully simulates the consistency checks carried out by each client C_i (which is a subset of the checks done by \mathcal{T} itself). If any of the simulated clients C_i catches an inconsistency, that client aborts it, as required by the protocol, and will prevent any client from proceeding to the next round. Whether a client aborts or not in a round can be perfectly simulated, but if none of the clients catch the inconsistency, then their behavior in the subsequent rounds depends on their private states. So, when none of the simulated clients have aborted after an inconsistency occurs in the messages reported on \mathbf{W}_j , \mathcal{T} continues the simulation as follows:
 - (a) \mathcal{T} first corrupts the server \overline{P}_j in the Π execution. Recall that Π is a non-erasing protocol, and hence \mathcal{T} gets to learn all the messages between \overline{P}_j and all the clients, as well as all the other servers. These messages can be used to reconstruct messages between the honest clients and (the simulated) \mathcal{G}_j .
 - (b) Then, in the execution of ρ_j^{OT} , \mathcal{T} directs \mathcal{S}_{ρ_j} to corrupt all the clients (as an adaptive corruption); in turn \mathcal{S}_{ρ_j} would request corruption of the clients in the ideal execution of \mathcal{G}_j , up on which \mathcal{T} provides the history of messages between \mathcal{G}_j and the clients (as obtained above) to \mathcal{S}_{ρ_j} .
 - (c) Given this, \mathcal{S}_{ρ_j} will provide \mathcal{T} with simulated current state for each of the clients in ρ_j^{OT} . (Note that it is not important whether ρ^{OT} is erasing or non-erasing.) Here we have required security of ρ^{OT} against two-step passive corruption: the simulator \mathcal{S}_{ρ_j} must be able to adaptively simulate the state of all the clients on the second-step corruption.
 - (d) \mathcal{T} continues the simulation using these states of the clients in ρ_j^{OT} . \mathcal{T} also has access to the further inputs that these programs receive from the clients as it has corrupted the server \overline{P}_j .

We need to argue that no environment can distinguish between \mathcal{T} running in an execution of Π and the actual execution of $\Phi_{\Pi, \rho}^{\text{OT}}$ with adversary \mathcal{A} . and that \mathcal{T} does not corrupt more than t

servers in Π (except with negligible probability).

The former follows by design of the simulation. In fact, if the simulation by \mathcal{S}_{ρ_j} were perfect, then \mathcal{T} also results in a perfect simulation. More generally, if the simulation by \mathcal{S}_{ρ_j} is indistinguishable (statistically or computationally), then the simulation by \mathcal{T} is also indistinguishable (statistically or computationally, respectively). We argue this using a few hybrids.

Firstly, \mathcal{G}_j and \overline{P}_j are identical, except for the (cosmetic) difference that \mathcal{G}_j explicitly allows the adversary to control when it sends messages to the honest parties, whereas with \overline{P}_j , it is the communication channel between \overline{P}_j and the honest parties that gives such control to the adversary. So, in the first hybrid derived from the simulation, we replace \overline{P}_j in the external outer protocol with \mathcal{G}_j , for every j , without changing the environment's behavior.

Now, consider $n + 1$ hybrid experiments: in the j -th hybrid execution $\mathcal{S}_{\rho_1}, \dots, \mathcal{S}_{\rho_{j-1}}$ (along with the functionalities $\mathcal{G}_1, \dots, \mathcal{G}_n$ they interact with) are replaced by the sessions of the protocol ρ^{OT} . The j -th and $j + 1$ -st hybrids differ in one session of ρ^{OT} . In the latter hybrid, \mathcal{S}_{ρ_j} simulates the protocol as long as the adversary remains honest (but curious) in the simulated session. Further, if and when there is a deviation in the session ρ_j^{OT} , \mathcal{S}_{ρ_j} obtains the honest clients' states in their interaction with \mathcal{G}_j (recall that \mathcal{T} detects the deviation, corrupts \overline{P}_j if the simulation needs to be continued, learns the the honest clients' states in the session) and simulates the honest clients' states in ρ_j^{OT} adaptively. If the simulation continues after this point (i.e., the simulated protocol does not abort), then the j -th and $j + 1$ -st hybrids evolve identically (wherein the honest clients faithfully follow the protocol in ρ_j^{OT} starting from their current state). Since the probability of aborting is independent of the inputs to the honest clients (which is ensured by the use of random inputs to OT channels, as further discussed below), it is perfectly simulated by \mathcal{T} . Thus if an environment can distinguish between these two hybrids, there is an environment which can break the adaptive security guarantee given by \mathcal{S}_{ρ_j} .

Then, by the hybrid argument, the first hybrid (which is identical to the simulation, taking place in an outer protocol execution) and the $n + 1$ -st hybrid above, which is identical to the execution of the compiled protocol, are indistinguishable to the environment.

What remains to be shown is that in the simulated execution \mathcal{T} does not corrupt more than t servers in the outer protocol. \mathcal{T} corrupts a server when a consistency check fails but an honest client fails to detect the deviation. Inconsistency involves a corrupt client violating the consistency between the messages in ρ_j^{OT} and the messages it reports on \mathbf{W}_j (corresponding to some server \overline{P}_j). If an honest client has \overline{P}_j in its watchlist then it will catch the first kind of inconsistency with constant probability. Indeed, unless the inconsistency involves an input to or output from an OT channel, the client will catch the inconsistency with probability 1. If the inconsistency is in the inputs given to an OT channel or in the reported outputs from an OT channel, then \mathcal{T} will find it, but the honest client is guaranteed only probability $\frac{1}{2}$ of finding it: if the corrupt client feeds a different input as a sender in the $\binom{2}{1}$ OT channel, then with probability $\frac{1}{2}$ the honest client will pick up the altered input; if the corrupt client as a receiver in the OT channel feeds one choice bit to the channel, but reports the other over the watchlist broadcast channel, then it has only probability $\frac{1}{2}$ of being able to correctly report the output it received from the channel (or even less, if OT is a string OT channel). Note that here we rely on the fact that all uses of OT in ρ_j^{OT} have random inputs from both parties.

So a server corruption by \mathcal{T} occurs with at most probability $1/2$, or only when no honest client has the relevant server in its watchlist. As long as the simulated protocol has not aborted, each server is in the watchlist of an honest client with probability at least k/n (conditioned on

servers corresponding to previous corruptions not being in the watchlist). Thus the probability of \mathcal{T} having to carry out $t/2$ server corruptions during the simulations is at most $(1 - \frac{k}{n})^{t/2} = 2^{-\Omega(k)}$ since $t = \Theta(n)$. While initializing the watchlists \mathcal{T} could corrupt up to $t/2$ servers (except with negligible probability). Thus in all, except with negligible probability, \mathcal{T} corrupts at most t servers in the Π session. □

3.4 Extensions to the Compiler

3.4.1 On Adaptive Security

Above we assumed that the inner protocol ρ^{OT} is secure against static two-step corruptions, and Π is secure against static client corruptions (and up to t adaptive server corruptions). Then the compiled protocol $\Phi_{\Pi, \rho}^{\text{OT}}$ is secure against static corruptions. However, if ρ^{OT} is secure against adaptive corruptions, depending on the security of Π we can get $\Phi_{\Pi, \rho}^{\text{OT}}$ to be secure against adaptive corruptions.

- If Π is secure against an adversary who can adaptively corrupt up to $m - 1$ clients and up to t servers, then $\Phi_{\Pi, \rho}^{\text{OT}}$ is secure against adaptive corruption up to $m - 1$ clients. All known constant-round protocols are restricted to this type of adaptive security, unless honest parties are allowed to erase data.
- If Π is secure against an adversary which could in addition, after the protocol execution ends, corrupt all the remaining honest clients and servers together, then $\Phi_{\Pi, \rho}^{\text{OT}}$ is secure against adaptive corruption of up to all m clients. This is the typical adaptive security feature of outer protocols whose round complexity depends on the circuit depth, and even of constant-round protocols if data erasure is allowed.

3.4.2 Removing restrictions on Π

We assumed that in the outer protocol the servers do not directly communicate with each other. This restriction can be removed, in fact in two ways. Firstly there is a simple modification to our compiler (and the security analysis) to allow such outer protocols. Alternately, we can pre-compile the outer protocol so that the server-to-server communications are securely and efficiently routed through the clients.

Modifying the compiler. First, we redefine the inner functionality \mathcal{G}_j so that any direct communication between two servers is implemented using (simple additive) secret-sharing among the clients. More precisely \mathcal{G}_j works as follows:

- \mathcal{G}_j internally runs the program for \overline{P}_j , which expects to interact with the clients \overline{C}_i and servers \overline{P}_j in the outer protocol Π .
- Recall that at the beginning of a round the program for \overline{P}_j accepts message $\overline{w}_{i \rightarrow j}$ from each client \overline{C}_i and message $\overline{u}_{j' \rightarrow j}$ from each server $\overline{P}_{j'}$, that were sent at the end of the previous round.

Instead, from each client C_i , \mathcal{G}_j accepts $(\bar{w}_{i \rightarrow j}, \bar{u}_{1 \rightarrow j}^{(i)}, \dots, \bar{u}_{n \rightarrow j}^{(i)})$; it reconstructs $\bar{u}_{j' \rightarrow j} = \bar{u}_{j' \rightarrow j}^{(1)} + \dots + \bar{u}_{j' \rightarrow j}^{(m)}$ for each $j' = 1, \dots, n$.⁹

It passes $(\bar{\mathbf{w}}_{\rightarrow j}, \bar{\mathbf{u}}_{\rightarrow j})$ to the program for \bar{P}_j , with $\bar{\mathbf{w}}_{\rightarrow j} = (\bar{w}_{1 \rightarrow j}, \dots, \bar{w}_{m \rightarrow j})$ as messages sent to \bar{P}_j by the clients \bar{C}_i and $\bar{\mathbf{u}}_{\rightarrow j} = (\bar{u}_{1 \rightarrow j}, \dots, \bar{u}_{n \rightarrow j})$ as the messages sent to \bar{P}_j by the other servers.

- Recall that at the end of a round, the program for \bar{P}_j sends messages $\bar{m}'_{j \rightarrow i}$ to each client \bar{C}_i and $\bar{u}'_{j \rightarrow j'}$ to each other server $\bar{P}_{j'}$. Instead, \mathcal{G}_j sends $(\bar{m}'_{j \rightarrow i}, \bar{u}_{j \rightarrow 1}^{(i)}, \dots, \bar{u}_{j \rightarrow n}^{(i)})$ to each client C_i , where $\bar{u}_{j \rightarrow j'}^{(i)}$ are random values such that $\bar{u}_{j \rightarrow j'}^{(1)} + \dots + \bar{u}_{j \rightarrow j'}^{(m)} = \bar{u}'_{j \rightarrow j'}$.

At the beginning of each round, the inputs that C_i provides to ρ_j^{OT} include the message from \bar{C}_i to \bar{P}_j at that round, and the share of any messages from any server $\bar{P}_{j'}$ to \bar{P}_j that was received as output from $\rho_{j'}^{\text{OT}}$ in the previous round.

The purpose of the watchlists is now two-fold: in addition to forcing honest behavior *within* each instance of the inner protocol, now the watchlists are used to force that the clients honestly relay the shares of server-to-server messages *across* multiple instances ρ_j^{OT} .

- To enforce consistency between multiple instances of the inner protocol, each client 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 ρ_j^{OT} for \mathcal{G}_j .
- Then, the consistency checks are extended to check consistency of messages in *every pair of watchlists* \mathbf{W}_j and $\mathbf{W}_{j'}$: Recall that in ρ^{OT} one server $\bar{P}_{j'}$ may send a message to another server \bar{P}_j , and then in $\Phi_{\Pi, \rho}^{\text{OT}}$ updating the shared state of the server \bar{P}_j will involve each client including an output from $\rho_{j'}^{\text{OT}}$ as a subsequent input to ρ_j^{OT} . If an honest client C_i has servers \bar{P}_j and $\bar{P}_{j'}$ on its watchlist, then C_i should check that outputs from $\rho_{j'}^{\text{OT}}$ and inputs to ρ_j^{OT} that each client reports over $\mathbf{W}_{j'}$ and \mathbf{W}_j are consistent with each other.

The simulation is then modified as follows. Firstly, the consistency checks done by the simulator \mathcal{T} are extended to include the above pairwise checks. Secondly, since \mathcal{G}_j 's interface with the clients is slightly differently from that of \bar{P}_j (as it accepts and gives out shares of server-to-server messages), the simulator \mathcal{T} translates between these two interfaces. The input to \mathcal{G}_j from a corrupt client C_i is of the form $(\bar{w}_{i \rightarrow j}, \bar{u}_{1 \rightarrow j}^{(i)}, \dots, \bar{u}_{n \rightarrow j}^{(i)})$ from which \mathcal{T} forwards $\bar{w}_{i \rightarrow j}$ to \bar{P}_j (after consistency checks). In response, \mathcal{T} obtains the message $\bar{m}'_{j \rightarrow i}$ from \bar{P}_j . Recall that \mathcal{S}_{ρ_j} expects a response from \mathcal{G}_j to C_i , which is of the form $(\bar{m}'_{j \rightarrow i}, \bar{u}_{j \rightarrow 1}^{(i)}, \dots, \bar{u}_{j \rightarrow n}^{(i)})$ where $\bar{u}_{j \rightarrow j'}^{(i)}$ are shares of server-to-server messages. As long as at least one client is uncorrupted, \mathcal{T} simply picks random values as the shares $\bar{u}_{j \rightarrow j'}^{(i)}$, and passes on $(\bar{m}'_{j \rightarrow i}, \bar{u}_{j \rightarrow 1}^{(i)}, \dots, \bar{u}_{j \rightarrow n}^{(i)})$ as the response from \mathcal{G}_j . (If adaptive corruption of clients is being considered (see Section 3.4.1), then on corrupting the last client, all past server-to-server messages are computed and the shares received by this last client is set to be consistent with these messages.) The only other change in \mathcal{T} is that it carries out the extra consistency checks, and if it

⁹By default, the addition refers to bitwise XOR. However, if the server-to-server message in the outer protocol consists of elements which belong to (and will undergo computations in) some other finite abelian group, then it may be better to use that group for the additive sharing of those elements. This allows implementing group additions performed by the server in the outer protocol non-interactively, by having clients directly add their shares.

detects an uncaught inconsistency between the messages in \mathbf{W}_j and $\mathbf{W}_{j'}$, then it corrupts one of the two servers \overline{P}_j and $\overline{P}_{j'}$.

The security analysis goes through as before (with a somewhat more extensive hybrid argument to handle the simulation of the server-to-server messages), but with the probability of corrupting t servers bounded by $2^{-\Omega(k/m^2)}$. This is because the upperbound on the probability of an uncaught inconsistency occurring is now $(1 - \frac{k(k-1)}{n(n-1)})$, corresponding to at least one of two servers between which an inconsistency occurs being not on the watchlist. Hence $(1 - \frac{k(k-1)}{n(n-1)})^{t/2} = 2^{-\Omega(k^2/n)} = 2^{-\Omega(k/m^2)}$, since $n = \Theta(km^2)$.

Precompiling the outer protocol. An alternate approach to handling an outer protocol with server-to-server messages is to first precompile it to one without such messages (while still remaining secure, with at most a constant factor reduction in the number of server corruptions that can be tolerated). The idea for this transformation is simple: instead of \overline{P}_j sending a message to $\overline{P}_{j'}$, it sends “shares” of this message to each client, who then forwards it to $\overline{P}_{j'}$ in the next round, who reconstructs the original message from the shares. The shares are derived using a secret sharing scheme which is also “non-malleable” — i.e., the corrupt clients cannot modify their shares without being caught cheating by $\overline{P}_{j'}$ while it tries to reconstruct the message from the shares. For this, we define a non-malleable secret-sharing scheme. As it turns out, this notion is closely related to (but slightly different from) the notion of Algebraic Manipulation Codes introduced by Cramer et al [15]. We shall focus on the 2-client setting, which requires a 2-party sharing scheme.

Definition 3.1 *A 2-party c -secure non-malleable secret sharing scheme over a (constant sized) alphabet Σ consists of two efficient algorithms, `share` (randomized) and `reconstruct` (deterministic), such that the following properties hold:*

- *It forms a secret-sharing scheme, i.e., each of the two shares produced by `share` is by itself distributed independently of the secret being shared, but when both shares are given to `reconstruct` it outputs the original secret.*
- *There exists a constant $c > 0$ such that for every (computationally unbounded) adversary \mathcal{A} , and every secret $x \in \Sigma$,*

$$\Pr[(\alpha, \beta) \leftarrow \text{share}(x), \alpha' \leftarrow \mathcal{A}(\alpha), \alpha' \neq \alpha, \text{reconstruct}(\alpha', \beta) \neq \perp] < 1 - c$$

- *There exist efficient probabilistic “faking” algorithms `Fake1` and `Fake2` such that `Fake1` outputs α, σ such that for any secret $x \in \Sigma$, we have that $(\alpha, \text{Fake}_2(\sigma, x))$ is distributed identically to the distribution `share`(x).*

We note that this definition of 2-party c -secure non-malleable secret sharing parallelizes in a natural way (where a reconstructed string is considered \perp if a \perp occurs anywhere inside the string), and so we can also talk of using such a scheme on arbitrarily long strings over the alphabet Σ .

We will be interested in constructions of such schemes where the shares themselves are constant-sized (depending on the constant c). We now describe a simple construction that achieves this definition, and yields only a constant-factor overhead. Let `MAC` be a family of constant-secure one-time message authentication code (MAC) schemes for producing MAC tags for constant-size messages, with the additional property of *non-redundant keys* — that is, that there do not exist

two keys K_1 and K_2 such that for all messages x , we have that $\text{MAC}_{K_1}(x) = \text{MAC}_{K_2}(x)$. Note that the the following simple MAC, $\text{MAC}_{a,b}(x) = ax + b$ has the properties that we seek, and in fact no two keys agree on more than one message x .

Now, to share a bit x , we first randomly additively share x into two bits x_1 and x_2 . Next¹⁰, we choose two MAC keys K_1, K_2 for MAC'ing bits, and we choose two more MAC keys K'_1, K'_2 for MAC'ing messages of length $|K_1|$. Now, the shares will be:

$$\alpha := \left((x_1, \text{MAC}_{K_2}(x_1)), (K_1, \text{MAC}_{K'_2}(K_1)), K'_1 \right), \quad \beta := \left((x_2, \text{MAC}_{K_1}(x_2)), (K_2, \text{MAC}_{K'_1}(K_2)), K'_2 \right)$$

The reconstruction checks for the natural consistency conditions, and outputs $x = x_1 + x_2$ if and only if they hold. It is easily verified that the above construction meets the non-malleability requirement for some constant c . Furthermore, using $\text{MAC}_{a,b}(x) = ax + b$ over a large enough field this construction can be used to yield a non-malleable secret sharing scheme achieving any level of security s , with an $O(\log(1/s))$ expansion of shares. More efficient schemes are possible, but this suffices for our purposes here.

This definition and construction can also be generalized to the multi-party case in a straightforward way.

It is easy to verify that if an outer protocol is modified so that server-to-server messages are routed through clients using a c -secure non-malleable secret-sharing scheme (for a suitable constant c), the resulting protocol still remains secure with a threshold say $t' = t/2$.

3.4.3 Using Inner Protocol With Only Static Passive Security

Using inner protocol secure against 2-step corruption. Note that our simulator \mathcal{T} relied on the security of the inner protocol ρ^{OT} against adaptive (passive) corruption in a limited manner: if an uncaught inconsistency occurred in ρ_j^{OT} , then \mathcal{T} got all the clients in ρ_j^{OT} to be corrupted together. So, it is sufficient that ρ^{OT} is secure only against “2-step passive corruption.” In this model of corruption, apart from some parties being passively corrupted at the beginning of the protocol, *all remaining parties can be corrupted together* at any point in the protocol. That is, a simulator \mathcal{S}_ρ should be able to simulate the protocol messages from the initially uncorrupted parties, and if at any point the adversary corrupts all the parties, then it should be able to simulate their internal states. At this point \mathcal{S}_ρ can corrupt all these parties in the ideal execution and is allowed to see all their past communication (inputs and outputs) with the ideal functionality.

A non-reactive inner functionality. Instead of using the reactive inner functionality \mathcal{G} , we can employ a non-reactive inner functionality $\hat{\mathcal{G}}$, which essentially carries out the functionality of \mathcal{G} one round at a time. This will be convenient in describing our modified simulation below, as we shall be simulating each round of the inner protocol separately. $\hat{\mathcal{G}}_j$ is described below, in terms of \mathcal{G}_j .

¹⁰Note that the simpler construction of $\alpha := (x_1, \text{MAC}_{K_2}(x_1), K_1)$ and $\beta := (x_2, \text{MAC}_{K_1}(x_2), K_2)$ does not satisfy the non-malleability requirement. Since the adversary may know x , given $\alpha = (x_1, \text{MAC}_{K_2}(x_1), K_1)$, it can find x_2 , and create $\alpha' = (x_1, \text{MAC}_{K_2}(x_1), K'_1)$, such that $\text{MAC}_{K_1}(x_2) = \text{MAC}_{K'_1}(x_2)$. Note that such malleability, in particular, leads to a “chosen-share attack” which can leak some information about the shared secret (if the adversary can learn whether or not its attack succeeded). While we do not rule out that some information could be learned in our definition of non-malleable secret sharing (indeed, our definition only asks for constant security), we do require that in any such attack, the adversary always has some constant probability of getting caught.

- From each client C_i , get input $(\bar{w}_{i \rightarrow j}, \bar{\Sigma}_j^{(i)})$, where $\bar{w}_{i \rightarrow j}$ will be considered the input of C_i to \mathcal{G}_j in a particular round, and $\bar{\Sigma}_j^{(i)}$ will be considered an additive share of the internal state of \mathcal{G}_j at the end of the previous round.¹¹
- Compute $\bar{\Sigma}_j^{(1)} + \dots + \bar{\Sigma}_j^{(m)}$, to reconstruct the internal state of \mathcal{G}_j at the end of previous round. Use the reconstructed state and the inputs $\bar{w}_{i \rightarrow j}$ (for $i = 1, \dots, m$) to evaluate the round function of \mathcal{G}_j , to obtain $(\bar{m}_{j \rightarrow 1}, \dots, \bar{m}_{j \rightarrow m}, \bar{\Sigma}'_j)$.
- To each client C_i give output $(\bar{m}_{j \rightarrow i}, \bar{\Sigma}'_j{}^{(i)})$ where $\bar{\Sigma}'_j{}^{(i)}$ form a random additive sharing of the updated state $\bar{\Sigma}'_j$.

Then our compiler will require an inner protocol, $\hat{\rho}^{\text{OT}}$ which securely realizes $\hat{\mathcal{G}}$ against 2-step passive corruption. The compiler's description remains virtually unchanged, but using $\hat{\rho}^{\text{OT}}$ in each round, with each client C_i feeding the output $\bar{\Sigma}'_j{}^{(i)}$ from one round as the input $\bar{\Sigma}_j^{(i)}$ in the next round. Correspondingly, the consistency checks are extended to ensure that the inputs to $\hat{\rho}^{\text{OT}}$ announced over the watchlist channel satisfy the requirement that the share $\bar{\Sigma}_j^{(i)}$ input by C_i in one round is indeed equal to the share $\bar{\Sigma}'_j{}^{(i)}$ (as declared over the watchlist channel) that is obtained by C_i in the previous round. The proof of security of the compiled protocol closely follows the previous proof with the following changes: \mathcal{T} now uses the simulator $\mathcal{S}_{\hat{\rho}}$ in each round, instead of \mathcal{S}_{ρ} ; also, since there is a new consistency check for each j , \mathcal{T} corrupts the server \bar{P}_j if this check fails. The rest of the simulation and proof remains virtually unchanged.

Using inner protocol with only static security. Earlier we required the inner protocol $\hat{\rho}^{\text{OT}}$ to be a little more than being secure against static passive corruption, namely secure against two-step passive corruption. This condition is indeed satisfied by the typical protocols which are secure against passive corruption. Nevertheless, as we show here, this requirement can in fact be removed, thanks to the fact that the $\hat{\rho}^{\text{OT}}$ protocol is used to emulate the server computation in an outer protocol Π , and Π is secure against adaptive corruption of the servers. However this modification requires us to (partially) sequentialize the activation of the servers, rather than let the clients interact with all the servers concurrently. We sketch this modification below.

For clarity of exposition, we shall *fully* sequentialize the activation of the servers in Π : at each round the clients will send messages to a single server and receive responses from it. Correspondingly, in the compiled protocol sessions of $\hat{\rho}^{\text{OT}}$ are run sequentially.

Earlier we gave a simulator \mathcal{T} which showed that no matter what the outer protocol is, the compiled protocol is as secure as the outer protocol with a limited number of server corruptions. But now we will need to use the security properties of the outer protocol; so we cannot give a simulator \mathcal{T} as before. Instead we give a simulator which works similar to \mathcal{T} but also uses the simulator for the outer protocol \mathcal{S}_{Π} . In fact, the new simulator does not directly use $\mathcal{S}_{\hat{\rho}}$ to carry out the simulation (though the existence of $\mathcal{S}_{\hat{\rho}}$ is necessary for the simulation to be indistinguishable).

The simulator works similar to \mathcal{T} , but simulating the outer protocol using \mathcal{S}_{Π} (while taking part in an ideal execution involving the functionality \mathcal{F}). In the (simulated) outer protocol, after sending inputs to a server \bar{P}_j and after receiving the response from it, the simulator “tentatively”

¹¹The additive sharing could be done in any suitable group. See Footnote ??.

corrupts that server: that is it requests \mathcal{S}_Π to provide the state of the server \overline{P}_j .¹² From this state it reconstructs the state of the clients in ρ_j^{OT} and carries out a faithful execution of ρ_j^{OT} . If at any point in this execution, an inconsistency is detected (and the execution is not aborted), then the simulator continues the simulation normally, because it already has the states of the clients with it. But if no inconsistency is detected, then the simulator must “undo” the corruption of \overline{P}_j . Note that as far as the execution in Π and \mathcal{S}_Π are concerned, the execution has not progressed while ρ_j^{OT} is being executed. So the simulator can continue \mathcal{S}_Π from the point just before requesting corruption of \overline{P}_j . This has the effect that the further simulation by \mathcal{S}_Π need not be consistent with the state provided by \mathcal{S}_Π during the tentative corruption. However the simulation is indeed consistent with the responses from \overline{P}_j that were provided just before the tentative corruption. Key to arguing the indistinguishability of this simulation is that, since ρ^{OT} is secure against passive corruption, the faithful execution of ρ_j^{OT} that the simulator carried out is indistinguishable from a simulation by $\mathcal{S}_{\hat{\rho}}$ using just the inputs and outputs to the server \overline{P}_j .

CONTINUE HERE

4 Instantiating the Building Blocks

For concrete applications of our compiler, we need to choose outer and inner protocols to which the compiler can be applied. The requirements on these components can be considered much easier to meet than security against active corruption in the case of no honest majority. As such the literature provides a wide array of choices that we can readily exploit.

Instances of the Outer Protocol. For the purpose of feasibility results, the classical BGW protocol [5, 10] can be used as the outer protocol. But in our applications, we shall resort to two efficient variants obtained from more recent literature [19, 20].¹³

Using a combination of [20, 14] (as described below) a boolean circuit C of size s and depth d (with bounded fan-in) can be evaluated with a total communication complexity of $O(s) + \text{poly}(n, k, d, \log s)$ bits, where k is a statistical security parameter, for n servers and any constant number of clients.¹⁴ The protocol requires $O(d)$ rounds. For this protocol the only type II functions in the servers’ program (see Section 3.1) consist of evaluating multiplications in a finite field \mathbb{F} whose size is independent of the number of servers. (Here we do not consider linear functions over \mathbb{F} , which can be handled “for free” by the inner protocol provided that the servers’ states are additively shared over \mathbb{F} among the clients.) The total number of multiplications computed by all servers throughout the protocol execution is $O(s) + \text{poly}(n, d)$ (for any constant number of clients).

An MPC protocol as above can be obtained by combining a version of an MPC protocol from [20] with algebraic geometric secret sharing over fields of constant size [14].¹⁵ This combination directly

¹²If we do not sequentialize the activations of the servers, inconsistency in ρ_j^{OT} for any j will require us to corrupt all the servers $\overline{P}_{j'}$ that are activated concurrently with ρ_j^{OT} . Since the outer simulator cannot handle more than t corruptions over all, we cannot afford this.

¹³Efficiency aside, by using UC-secure outer protocols, our compiled protocols are also UC-secure.

¹⁴While 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 most natural instances of large circuits.

¹⁵Using Franklin and Yung’s variant of Shamir’s secret sharing scheme [46, 23], as originally done in [20], would result in logarithmic overhead to the communication complexity of the protocol, and a polylogarithmic overhead in the complexity of the applications.

yields a protocol with the above properties for \mathbf{NC}^0 circuits, which was recently used in [35] to obtain constant-rate zero-knowledge proofs and in [30] to obtain constant-rate OT combiners. In Appendix A we describe this construction in more detail, extended to handle arbitrary depth- d circuits, at the cost of requiring $O(d)$ rounds.

Another useful instance of an outer protocol is obtained from the *constant-round* protocol from [19], as described in Section 5.2. Unlike the previous constant-round MPC protocol from [4], this protocol only makes a black-box use of a pseudorandom generator.

Instances of the Inner Protocol. The main choice of the inner protocol, which suffices for most of our applications, is the simple version of the GMW protocol [25, 24] that provides perfect security against a *passive* adversary in the OT-*hybrid* model, and is easily seen to be secure against two-step passive corruption (Section 3.2) as well. The communication complexity is $O(m^2s)$ where m is the number of clients and s is the size of the boolean circuit being evaluated (excluding XOR gates). The round complexity is proportional to the circuit depth (where here again, XOR gates are given for free). When evaluating functions in \mathbf{NC}^1 (which will always be the case in our applications) the inner protocol can be implemented using a single round of OTs in the two-party case, or a constant number of rounds in the general case, without compromising unconditional security. This is done by using a suitable randomized encoding of the function being computed, e.g., one based on an unconditionally secure variant of Yao’s garbled circuit technique [47, 33]. In the two-party case, the protocol needs to use only as many OTs as the length of the *shorter* input. This will be useful for some applications.

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.

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 the constant-rate MPC protocol described in Section 4. The inner protocol can be taken to be the “passive-secure GMW” protocol in the OT-hybrid model.

Theorem 2 *Let C be a boolean circuit of size s , depth d and constant fan-in representing an m -party deterministic 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 $O(s) + \text{poly}(k, d, \log s)$, 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 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 $\text{poly}(m)$. The theorem applies also to *reactive* functionalities, by naturally extending the outer protocol to this case, and

to *randomized* functionalities, provided that they are adaptively well-formed [12] or alternatively if honest parties are trusted to erase data.

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 [25, 24], 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 [36, 18] (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 [19], providing a black-box alternative to a previous protocol of Beaver, Micali, and Rogaway [4] that made a non-black-box use of a pseudorandom generator. The case of two-party computation was recently resolved by Lindell and Pinkas [38] (see also [40, 37]), 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 [38] does not seem to easily extend to more than two parties. Partial progress in this direction was recently made in [28].

By applying our compiler to a variant of the MPC protocol from [19], 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 [38] 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 [19] 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 [19] 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 any number of clients m . In any case where a server in the protocol of [19] 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 I, 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 [19] (that only relies on the semantic security of E) applies as is. \square

5.3 OT Extension in the Malicious Model

Beaver [3] 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 [32]. 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”¹⁶ 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, [32] employed a cut-and-choose approach which multiplies the complexity by a statistical security parameter. A partial improvement was recently given in [30], 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 [31]. 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 [30] (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 [32]. 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 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

¹⁶ The correlation robustness property defined in [32] 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.

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 II 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 suitable randomized encoding (see Section 4), 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 [32]. 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 $O(k^2)$ OTs (which also includes the OTs 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 [32]) 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 [34] and Haitner [29], 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 [31] allows one to 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 [30], 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] B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.
- [2] D. Beaver. Precomputing oblivious transfer. In D. Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 1995.
- [3] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proc. 28th STOC*, pages 479–488. ACM, 1996.
- [4] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM, 1990.
- [5] 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.
- [6] P. Bogetoft, D. L. Christensen, I. Damgård, 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/>.
- [7] G. Bracha. An $o(\log n)$ expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.
- [8] G. Brassard, C. Crépeau, and M. Santha. Oblivious transfers and intersecting codes. *IEEE Transactions on Information Theory*, 42(6):1769–1780, 1996.
- [9] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.
- [10] 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” available at the ECCC archive TR01-016. Extended abstract in FOCS 2001.
- [11] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Revised version of [10].
- [12] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party computation. In *Proc. 34th STOC*, pages 494–503. ACM, 2002.
- [13] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th STOC*, pages 11–19. ACM, 1988.

- [14] 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.
- [15] R. Cramer, Y. Dodis, S. Fehr, C. Padró, and D. Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. Cryptology ePrint Archive, Report 2008/030; To appear in Eurocrypt 2008, 2008. <http://eprint.iacr.org/2008/030>.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudo-random generator. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2005.
- [20] 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.
- [21] 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.
- [22] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
- [23] M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710. ACM, 1992.
- [24] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.
- [25] 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 [24, Chap. 7] for more details.
- [26] O. Goldreich and R. Vainish. How to solve any protocol problem - an efficiency improvement. In C. Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 1987.
- [27] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *DISC*, volume 2508 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2002.

- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.
- [34] Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. Black-box constructions for secure computation. In *STOC*, pages 99–108. ACM, 2006.
- [35] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30. ACM, 2007.
- [36] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31. ACM, 1988.
- [37] 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.
- [38] 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.
- [39] U. M. Maurer. Secure multi-party computation made simple. In S. Cimato, C. Galdi, and G. Persiano, editors, *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 14–28. Springer, 2002.
- [40] 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.
- [41] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. In *STOC*, pages 213–223. ACM, 1990.

- [42] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.
- [43] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2000.
- [44] M. Prabhakaran. *New Notions of Security*. PhD thesis, Department of Computer Science, Princeton University, 2005.
- [45] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.
- [46] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), Nov. 1979.
- [47] A. C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE, 1986.

A A Constant-Rate Outer MPC Protocol

In this section we describe the construction of the main instance of the outer MPC protocol Π (see Section 3.1), which we use to derive most of our applications. (The only application which does not rely on this outer protocol is the one described in Section 5.2). The protocol involves n servers and m clients, where only clients have inputs and outputs. Before describing the construction, we summarize the efficiency achieved by this protocol. For simplicity we shall restrict ourselves to $m = O(1)$ and $n = \text{poly}(k)$, where k is a statistical parameter. To evaluate a boolean circuit C of size s and depth d (with bounded fan-in), the communication complexity is $C_\Pi = O(s) + \text{poly}(k, d, \log s)$ bits.¹⁷ Assuming broadcast as an atomic primitive, the protocol requires $r_\Pi = O(d)$ rounds.

Recall that the complexity of the inner protocol ρ^{OT} depends on the functionality \mathcal{G} that it implements. As described in Section 3.2, since ρ^{OT} needs to be secure only against passive corruptions, it is the complexity of type II computations by the servers in Π that will affect the complexity of ρ^{OT} . The total complexity of such computations (say, the number of finite functions evaluated) by all servers throughout the entire protocol execution of Π is $O(s) + \text{poly}(n, d)$. In other words, the amortized complexity of type II computations in \mathcal{G} , per round, per server is $O(\frac{s}{nr_\Pi}) + \text{poly}(n, d)$, and (using the passive-secure GMW protocol) this is the amortized communication complexity of ρ^{OT} . That is, in Theorem 1 the term $nr_\Pi C_\rho$ can be replaced by $O(s) + \text{poly}(n, d)$.

The MPC protocol naturally extends an MPC protocol from [20], combined with Algebraic-Geometric secret sharing over fields of constant size [14]. This combination yields a protocol with the above properties for \mathbf{NC}^0 circuits, which was recently used in [35] to obtain constant-rate zero-knowledge proofs and in [30] to obtain constant-rate OT combiners. Here we present a natural extension of this protocol that applies to arbitrary circuits, at the cost of requiring $O(d)$ rounds.

To simplify the following exposition we will only consider the case of two clients Alice and Bob. An extension to the case of a larger number of clients is straightforward.

Another simplifying assumption is that the circuit C consists of d layers of width w , where each layer performs NAND operations on values produced by the previous layer only. Circuits of an arbitrary structure can be easily handled at a worst-case additional cost of $\text{poly}(n, d)$. (This cost can be amortized away for almost any natural instance of a big circuit. For instance, a sufficient condition for eliminating this cost is that for any two connected layers there are at least n wires connecting between the layers.)

As a final simplification, we present the protocol in terms of Shamir’s secret sharing scheme, in which the field size is small but non-constant. The claimed efficiency is obtained by replacing Shamir’s scheme with an analogous scheme based on AG codes, described in [14], in which the field size is independent of the number of players. (Using Shamir’s scheme would result in a logarithmic overhead to the communication complexity of the protocol, and a polylogarithmic overhead in the complexity of the applications.) This change results in an additional decrease of the fractional security threshold, that will not affect the asymptotic complexity of the protocol.

A.1 Building Blocks

Our protocol heavily relies on tools and sub-protocols from previous works that we describe below.

¹⁷While 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 .

Secret sharing for blocks. Shamir’s secret sharing scheme [46] distributes a secret $s \in F$ by picking a random degree- d polynomial p such that $p(0) = s$, and sending to server j the point $p(j)$. (Here F is a finite field such that $|F| > n$.) The generalization of Franklin and Yung [23] achieves far better efficiency with a minor cost to the security level. In this scheme, a *block* of ℓ secrets (s_1, \dots, s_ℓ) is shared by picking a random degree- d polynomial p such that $p(1 - j) = s_j$ for all j , and distributing to server j the point $p(j)$. (Here we assume that $-\ell + 1, \dots, n$ denote $n + \ell$ distinct field elements.) Any set of $d + 1$ servers can recover the entire block of secrets by interpolation. On the other hand, any set of $t = d - \ell + 1$ servers learn nothing about the block of secrets from their shares. (Secret sharing scheme in which there is a gap between the privacy and reconstruction thresholds are often referred to as “ramp schemes”.) For our purposes, we will choose ℓ to be a small constant fraction of n and d a slightly bigger constant fraction of n (for instance, one can choose $d = n/3$ and $\ell = n/4$). This makes the amortized overhead of distributing a field element constant, while maintaining secrecy against a constant fraction of the servers.

Adding and multiplying blocks. Addition and multiplication of shared blocks is analogous to the use of Shamir’s scheme in the BGW protocol [5]. Suppose that a block $a = (a_1, \dots, a_\ell)$ was shared via a polynomial p_a and a block $b = (b_1, \dots, b_\ell)$ was shared via a polynomial p_b . The servers can then locally compute shares of the polynomial $p_a + p_b$, which are valid shares for the sum $a + b$ of the two blocks. If each server multiplies its two local shares, the resulting n points are a valid secret-sharing using the degree- $(2d)$ polynomial $p = p_a p_b$ of the block $ab = (a_1 b_1, \dots, a_\ell b_\ell)$. Note, however, that even if p_a, p_b were obtained from a random secret sharing, $p_a p_b$ is not a random degree- $(2d)$ secret sharing of ab . Thus, if we want to reveal ab we will need to mask $p_a p_b$ by a random degree- $2d$ secret-sharing of a block of 0’s before revealing it. Also, in order to use ab for subsequent computations we will need to reduce its degree back to d .

Proving membership in a linear space. Our protocol will often require a client to distribute to the servers a vector $v = (v_1, \dots, v_n)$ (where each v_j includes one or more field elements) while assuring them that v belongs to some linear space L . This should be done while ensuring that the adversary does not learn more information about v than it is entitled to, and while ensuring the honest parties that the shares they end up with are consistent with L . For efficiency reasons, we settle for having the shares of the honest parties *close* to being consistent with L . Since we will only use this procedure with L that form an error correcting code whose minimal distance is a large constant multiple of d , the effect of few “incorrect” shares can be undone via error-correction. (In fact, in our setting of security with abort error detection will be sufficient.) More concretely, our procedure takes input $v = (v_1, \dots, v_n) \in L$ from a dealer D (Alice or Bob). In the presence of an active, adaptive adversary who may corrupt any client and at most t servers, it should have the following properties:

- **Completeness:** If D is uncorrupted then every honest server j outputs v_j .
- **Soundness:** Suppose D is corrupted. Except with negligible probability, either all honest servers reject (in which case the dealer is identified as being a cheater), or alternatively the joint outputs of all n servers are most $2t$ -far (in Hamming distance) from some vector in $v \in L$.
- **Zero-Knowledge:** If D is uncorrupted, the adversary’s view can be simulated from the shares v_j of corrupted servers.

Verifiable Secret Sharing (VSS) can be obtained by applying the above procedure on the linear space defined by the valid share vectors. Note that in contrast to standard VSS, we tolerate some inconsistencies to the shares on honest servers. As discussed above, such inconsistencies can be corrected by the higher level protocol.

Implementing proofs of membership. We will employ a sub-protocol from [20] (Protocol 5.1) for implementing the above primitive. This protocol amortizes the cost of proving that many vectors v^1, \dots, v^q owned by the same dealer D belong to the same linear space L by taking (ϵ -biased, pseudo-)random linear combinations of these vectors together with random vectors from L that are used for blinding. The high level structure of this protocol is as follows.

- *Distributing shares.* D distributes v^1, \dots, v^q to the servers.
- *Distributing blinding vectors.* D distributes random vectors $r_1, \dots, r_\kappa \in L$ that are used for blinding. (This step ensures the zero-knowledge property; soundness does not depend on the valid choice of these vectors r .)
- *Coin-flipping.* The servers flip coins that produce κ independent seeds for an ϵ -biased generator [41]. Each of these seeds defines a (pseudo-)random linear combination of the q vectors distributed by the dealer. Since both κ and the seed length are small ($\text{poly}(k, n, \log s)$) we can use an MPC protocol based on inefficient VSS to implement the coin flips. (Moreover, in the case of two clients we let the other client, who does not serve as a dealer, pick the coins and broadcast them.)
- *Proving.* The dealer computes the κ linear combinations of its vectors v^i defined by the coin tosses, and adds to each linear combination the corresponding blinding vector. It broadcasts the results. (Here too, the complexity is only ($\text{poly}(k, n, \log s)$)). We note that if the code defined by L is efficiently decodable from up to $2t$ errors (which will be the case for all L we will employ) this step can be skipped.
- *Complaining.* Each server applies the κ linear combinations to its part of the vectors distributed by the dealer, and ensures that the result is consistent with the values broadcast in the previous step. Also, each server checks that all κ vectors broadcast by the dealer are in L . If any of these checks fail for a server, it broadcasts a complaint.

This is the only step of the protocol that requires servers to compute functions whose domain is not finite. But as the inputs for this computation are known to the dealer, this is a type I computation. As such, its complexity will not affect the complexity of the inner protocol ρ^{OT} .

- *Outputs.* If more than t servers broadcast a complaint, all servers reject. Otherwise, the servers output the shares distributed by the dealer in the first step (discarding the blinding vectors and the results of the coin-flips).

We will sometimes employ the above protocol in a scenario where vectors v^1, \dots, v^q are already distributed between the servers and known to the dealer, and the dealer wants to convince the servers that these shares are close to being consistent with L . In such cases we will employ the above sub-protocol without the first step. We note that in any application of the above procedure in our protocol, if the dealer is caught cheating there are two ways to proceed (depending on the

type of required security): (1) abort; (2) eliminate the dealer, replacing his input with 0, and restarting the protocol without this client. (Restarting is necessary only in the case where there are more than 2 clients.) In any case, the event of a dealer being identified as a cheater can be easily incorporated into the simulation, and in particular is independent of the inputs of the uncorrupted client/s.

Proving global linear constraints. We will often need to deal with a more general situation of proving that vectors v^1, \dots, v^q not only lie in the same space L , but also satisfy additional global constraints. A typical scenario applies to the case where the v^i are shared blocks defined by degree- d polynomials. In such a case, we will need to prove that the secrets shared in these blocks satisfy a specified replication pattern (dictated by the structure of the circuit C we want to compute). Such a replication pattern specifies which entries in the q blocks should be equal. A key observation made in [20] is that: (1) such a global constraint can be broken into at most $q\ell$ atomic conditions of the type “entry i in block j should be equal to entry i' in block j' ”, and (2) by grouping these atomic conditions into ℓ^2 types defined by (i, i') , we can apply the previous verification procedure to simultaneously verify all conditions in the same type. That is, to verify all conditions of type (i, i') each server concatenates his two shares of every pair of blocks that should be compared in this type, and then applies the previous verification procedure with L being the linear space of points on degree- d polynomials (p_1, p_2) which satisfy the constraint $p_1(1 - i) = p_2(1 - i')$. Note that the communication required by this procedure will still be $\text{poly}(k, n, \log s)$, and the local computations done by the servers are still performed on inputs that are known to the dealer (and thus can be efficiently handled using the watch list approach). Unlike [20] we will also employ the above procedure in the case where p_1, p_2 may be polynomials of different degrees (e.g., d and $2d$), but the same technique applies to this more general case as well.

A.2 The Protocol

The protocol is a natural extension of the protocol from [20], which can be viewed as handling the special case of \mathbf{NC}^0 functions using a constant number of rounds. We handle circuits of depth d by using $O(d)$ rounds of interaction. The protocol from [20] handles general functions by first encoding them into \mathbf{NC}^0 functions (using [1]), but such an encoding step is too expensive for our purposes.

Recall that we assume the circuit C to consist of d layers of width w each, and that each gate in layer i depends on two outputs from from layer $i - 1$. To further simplify the exposition we assume that each such gate is the NAND of its two inputs. Unlike protocols based on Yao’s garbled circuit technique [47], the protocol can be easily extended to compute arithmetic circuits over F , which can be useful for many applications of secure computation.

The high level strategy is to pack the inputs for each layer into blocks in a way that allows to evaluate the NAND gates in this layer “in parallel” on pairs of blocks. That is, the computation of the layer will consist of disjoint parallel computations of the form a NAND b , where a and b are blocks of ℓ binary values and the NAND operation is performed coordinate-wise. This will require blocks to be set up so that certain inputs appear in several places. Such a replication pattern will be enforced using the procedure described above. Throughout the protocol, if a prover is caught cheating the protocol can be aborted or restarted as discussed above.

The protocol will proceed as follows:

1. *Sharing inputs.* The clients arrange their inputs into blocks with a replication pattern that sets up the parallel evaluation for the first layer (namely, so that the first layer will be evaluated by taking the NAND of blocks 1,2, of blocks 3,4, etc.). Each client then secret-shares its blocks, proving to the servers that the shares of each block agree with a polynomial of degree at most d and that the secrets in the shared blocks satisfy the replication pattern determined by the first layer of C . (Such proofs are described in the previous section.) In addition, in the case of boolean circuits servers need to be convinced that all secrets are either 0 or 1. This can be done in several standard ways. If F is of characteristic 2, then valid shares of 0/1 blocks form a linear space over the binary field, and therefore such proofs can be obtained using the procedures described above. In general, the servers can securely reveal $1 - a \cdot a$ for each block a (which should evaluate to a block of 0's) or alternatively work with an arithmetic circuit C that starts by powering the inputs to the order of F .
2. *Evaluating C on shared blocks.* The main part of the protocol is divided into d phases, one for evaluating each layer of C . For $h = 1, 2, \dots, d$ we repeat the following:

- *NANDing and blinding.* At the beginning of the phase, the inputs to layer h are arranged into blocks, so that the outputs of layer h can be obtained by taking the NAND of each consecutive pair of blocks. Moreover, each block is secret-shared using a degree- d polynomial. We would like to reveal the outputs of the layer to Alice, masked by random blinding blocks picked by Bob. For this, Bob will VSS random blocks, one for each block of output. The secret-sharing of these blocks is done using polynomials of degree $2d$.

(Again, verifying that the shares distributed by Bob are valid is done using the procedure described above.) For every pair of input blocks a, b whose NAND is computed, each server j locally computes the degree-2 function $c(j) = 1 - a(j)b(j) + r(j)$, where $a(j), b(j)$ are its shares of a, b and $r(j)$ is its share of the corresponding blinding block r distributed by Bob. For each pair of blocks combined in this way, the server sends his output (a single field element) to Alice. Note that the points $c(j)$ lie on a random degree- $2d$ polynomial p_c , and thus reveal no information about a, b . Moreover, the polynomial p_c can be viewed as some valid degree- $2d$ secret sharing of the block $c = 1 - ab + r$.

- *Reducing degree and rearranging blocks for layer $h + 1$.* Alice checks that the points $c(j)$ indeed lie on a polynomial p_c of degree at most $2d$ (otherwise she can either apply error correction or abort, depending on the security setting). Then she recovers the blinded output block $c = 1 - ab + r$ by letting $c_j = p_c(1 - j)$. Now Alice uses all blinded blocks c obtained in this way to set up the (blinded) blocks for computing layer $h + 1$.

For this, she sets up a new set of blocks that are obtained by applying a projection (namely, permuting and copying) to the blocks c that corresponds to the structure of layer $h + 1$. (In particular, the number of new blocks in which an entry in a block c will appear is precisely the fan-out of the corresponding wire in C .) Let c' denote the rearranged blinded blocks.

Now Alice secret-shares each block c' using a degree- d polynomial $p_{c'}$. She needs to prove to the servers that the shares she distributed are of degree d and that the entries of the shared blocks c' satisfy the required relation with respect to the blocks c that are already shared between the servers using degree- $2d$ polynomials. Such a proof can be efficiently carried out using the procedure described above. Note that pairs of polynomials $(p_c, p_{c'})$ such that p_c is of degree at most $2d$, $p_{c'}$ is of degree at most d , and $p_c(i) = p_{c'}(j)$ form

a linear space (for any fixed i, j), and hence the $2n$ evaluations of such polynomials on the points that correspond to the servers form a linear subspace of F^{2n} . Also, the corresponding code will have a large minimal distance because of the degree restriction, which ensures that the adversary cannot corrupt a valid codeword without being detected (and even corrected).

- *Unblinding.* To set up the input blocks for the evaluation of layer $h+1$, we need to cancel the effect of the blinding polynomials p_r distributed by Bob. For this, Bob distributes random degree- d unblinding polynomials $p_{r'}$ that encodes blocks r' obtained by applying to the r blocks the same projection defined by the structure of layer $h+1$ that was applied by Alice. Bob proves that the polynomials $p_{r'}$ are consistent with the p_r similarly to the corresponding proof of Alice in the previous step. (In fact, both sharing the $p_{r'}$ and proving their correctness could be done in the first step.) Finally, each server obtains its share of an input block a for layer $h+1$ by letting $a(j) = c'(j) - r'(j)$.
3. *Delivering outputs.* The outputs of C are revealed to the clients by having the servers send their shares of each output block to the client who should receive it. By the robustness of the secret sharing, the client can decode the correct output from the shares. (Alternatively, in our setting we can allow a client to abort if the shares it receives are not consistent with any degree- d polynomial).

The security of the above protocol can be proved along the lines of the BGW protocol [5] and the protocol from [20].

We finally note that when the circuit C is an \mathbf{NC}^0 circuit (namely, each bit of the output depends on a constant bits of the input) the above protocol can be implemented using a single phase, as was originally done in [20]. This variant of the protocol will be useful for the application to extending OTs.

B Non-Interactive 2-Party SFE

In this section we describe a minimally interactive two-party protocol in the OT-hybrid model that uses only a *single* round of OTs and no additional interaction. The SFE functionalities considered here provide output to only one party.

First, we shall describe a protocol that achieves security against covert-adversaries, but with a deterrence probability that can be easily reduced to negligible by choosing parameters appropriately. We describe the protocol in two parts.

B.1 Reducing Covert-adversary 2-Party SFE to Covert-adversary Certified-OT

We shall use an intermediate 2-party functionality $\mathcal{F}_{\text{cov-cOT}}$, for “covert-adversary certified-OT.”

Covert-adversary Certified-OT Functionality. Parameters of $\mathcal{F}_{\text{cov-cOT}}$ include a function C , the number of pairs of strings that are being transferred, m , the length of these strings, and a “deterrence probability” ϵ .

1. $\mathcal{F}_{\text{cov-cOT}}$ takes from the sender input $\Gamma = ((s_0^1, s_1^1), \dots, (s_0^m, s_1^m); w)$ that is m pairs of strings and a “witness” w . It takes from the receiver inputs c_1, \dots, c_m .

2. If the sender is corrupt, it allows the sender to send a command `cheat` also. In this case, with probability ϵ , $\mathcal{F}_{\text{cov-cOT}}$ will produce the message `corrupted` as output to both the parties and terminates, and with probability $1 - \epsilon$, will (a) give c_1, \dots, c_m to the sender and (b) allow the sender to specify an output for the receiver.
3. If the sender does not include the `cheat` command in the input, then the receiver gets $(s_{c_1}^1, \dots, s_{c_m}^m; C(\Gamma))$.

Given a decomposable randomized encoding h for a function g , it is fairly straight forward to use a simple generalization of Yao's protocol for secure 2-party evaluation of g in the certified-OT-hybrid model. In the $\mathcal{F}_{\text{cov-cOT}}$ -hybrid model, this protocol is a secure realization of covert-adversary SFE of g .

Let g 's input be in two parts: A 's input a and B 's input b . Recall that a decomposable randomized encoding of g can be written as $h(x, r) = \{h_i(x_i, r)\}_{i=1}^{|x|}$, where x is the input to g . Since $x = (a, b)$, we will rewrite this as $h(a, b, r) = \{h_i^A(a_i, r)\}_{i=1}^{|a|} \circ \{h_i^B(b_i, r)\}_{i=1}^{|b|}$.

The function C associated with $\mathcal{F}_{\text{cov-cOT}}$ is defined as

$$C(\{(s_0^i, s_1^i)\}_{i=1}^{|b|}; (a, r)) = (\{h_i^A(a_i, r)\}_{i=1}^{|a|}; R(\{(s_0^i, s_1^i)\}_{i=1}^{|b|}, r)),$$

where R is a predicate which checks that $\{(s_0^i, s_1^i)\}_{i=1}^{|b|} = \{(h_i^B(0, r), h_i^B(1, r))\}_{i=1}^{|b|}$.

1. A picks a random string r (for the randomized encoding of g) and prepares the following input for $\mathcal{F}_{\text{cov-cOT}}$ (with the associated function C described above): $(\{(h_i^B(0, r), h_i^B(1, r))\}_{i=1}^{|b|}; (a, r))$.
2. B inputs $(b_1, \dots, b_{|b|})$ to $\mathcal{F}_{\text{cov-cOT}}$, and obtains $(\{h_i^B(b_i, r)\}_{i=1}^{|b|}; \{h_i^A(a_i, r)\}_{i=1}^{|a|}; z)$. B aborts if $z \neq 1$.
3. B computes $g(a, b)$ from $h(a, b, r) = \{h_i^A(a_i, r)\}_{i=1}^{|a|} \circ \{h_i^B(b_i, r)\}_{i=1}^{|b|}$, and outputs it.

Proof of Security. If A is corrupt, the simulation is straight forward: the simulator obtains her inputs to $\mathcal{F}_{\text{cov-cOT}}$, computes C and checks if the predicate R evaluates to 1 on this input. If so, it sends a to $\mathcal{F}_{\text{cov-g}}$. If A sends a `cheat` command to $\mathcal{F}_{\text{cov-cOT}}$, then the simulator also sends a `cheat` command to $\mathcal{F}_{\text{cov-g}}$. If $\mathcal{F}_{\text{cov-g}}$ responds with the inputs of B , then the simulator sends these to A as the response from $\mathcal{F}_{\text{cov-cOT}}$. Further, in this case, A sets the outputs of $\mathcal{F}_{\text{cov-cOT}}$, which are received by the simulator, who uses it to carry out the rest of the protocol of B ; the simulator will then instruct $\mathcal{F}_{\text{cov-g}}$ to output whatever this simulated B outputs. It is easily seen that this is a perfect simulation if $\mathcal{F}_{\text{cov-g}}$ has the same deterrence probability as $\mathcal{F}_{\text{cov-cOT}}$.

If B is corrupt, then also there is a simple simulation, which depends on the privacy property of the randomized encoding. The simulator obtains the bits of B 's input from $\mathcal{F}_{\text{cov-cOT}}$, sends it to $\mathcal{F}_{\text{cov-g}}$ and obtains the output for B . It then constructs a random encoding consistent with this output value and B 's inputs. This is used to prepare a simulated output from $\mathcal{F}_{\text{cov-cOT}}$.

(Note that $\mathcal{F}_{\text{cov-cOT}}$ does not allow B to send a `cheat` message.)

B.2 Reducing Covert-adversary Certified-OT to OT

In this section we give a two-party protocol cOT^{OT} in the OT-hybrid model, which achieves the "covert-adversary" Certified-OT functionality $\mathcal{F}_{\text{cov-cOT}}$. Our protocol is built by compiling an MPC protocol, η (involving more than two parties, with a certain level of information theoretic security against passive corruption) into a two-party protocol in the OT-hybrid model.

Protocol η . First we describe the requisite properties of the protocol η .

- **Participants:** There are 2 *input clients*, q *servers* and $2Lm + 1$ *output clients* for some $L > 1$. (Here q will be a constant, set to 3 or 5, for instance. m is the number of pairs of strings that the sender wants to send in the certified OT functionality provided by the compiled protocol. L can be set to 2 to get a deterrence value of $1/2$, or if a higher deterrence value is needed, a higher constant.) We denote the 2 input clients by I_0 and I_1 ; we denote the $2Lm + 1$ output clients by $Z_{\ell 0}^i$ and $Z_{\ell 1}^i$, (for $i = 1, \dots, m$ and $\ell = 1, \dots, L$) and Z^0 .
- **Functionality:** We define the following functionality \mathcal{H} . Let x_0 and x_1 denote the inputs of I_0 and I_1 . \mathcal{H} parses $x_0 \oplus x_1$ as m pairs of strings $((s_0^1, s_1^1), \dots, (s_0^m, s_1^m))$ and a “witness” w . Z^0 is given the function $C((s_0^1, s_1^1), \dots, (s_0^m, s_1^m); w)$. For each i , $Z_{\ell 0}^i$ and $Z_{\ell 1}^i$ ($\ell = 1, \dots, L$) receive random strings $z_{\ell 0}^i$ and $z_{\ell 1}^i$ subject to the constraint that $\bigoplus_{\ell} z_{\ell r_\ell}^i = s_{\bigoplus_{\ell} r_\ell}^i$ for all $r \in \{0, 1\}^L$. That is, $z_{\ell b}^i$ are random such that $\bigoplus_{\ell} z_{\ell 0}^i = s_0^i$ and $z_{\ell 1}^i = z_{\ell 0}^i \oplus s_0^i \oplus s_1^i$.
- **Security:** η must be t_0 -private, for some $t_0 \geq 2$. More precisely, it securely realizes the functionality \mathcal{H} against passive (honest-but-curious), adversaries who can corrupt up to t_0 servers, and any number of input and output clients. The security is perfect.
- **Structure of the protocol:** We will require that the input clients talk only to the servers and that output clients only receive messages and never send messages. We can allow η to be in the OT-hybrid model.
- **Complexity:** We require the communication complexity of the protocol to be linear in the circuit size of C .

Standard MPC protocols from the literature can be easily adapted to obtain a protocol η that fits the above requirements. So, for instance, for 2-security (i.e., $t_0 = 2$), we can let $q = 3$ and use the (semi-honest) GMW protocol [25] in an OT-hybrid model, or let $q = 5$ and use the BGW protocol [5] (or a simpler protocol due to Maurer [39]).

Protocol ϕ . Similar to η we also need a (simpler) protocol for an “equality check,” with a similar protocol structure and security guarantee. ϕ has 4 input clients and one output client, and q servers without input or output. ϕ (stand-alone) securely realizes the following functionality \mathcal{E} , against $t_0 \geq 2$ passive server corruptions. The security is perfect.

Let the input clients be I_0, I_1, I'_0 and I'_1 , with inputs x_0, x_1, x'_0 and x'_1 , respectively. Then \mathcal{E} outputs 1 to the output client if and only if $x_0 \oplus x_1 = x'_0 \oplus x'_1$.

Protocol cOT^{OT} . The certified-OT protocol cOT^{OT} proceeds as follows in the OT-hybrid model. Let κ be a statistical security parameter. (Later we will set κ to be a constant, independent of the final security parameter, m and the final circuit size.)

- *Run “MPC in the head”:* The sender prepares κ total views of the execution of the protocol η and $\binom{\kappa}{2}$ total views of the execution of the protocol ϕ . We will refer to the κ executions of η as η_j ($j = 1, \dots, \kappa$) and the $\binom{\kappa}{2}$ executions of ϕ as $\phi_{jj'}$ (for $1 \leq j < j' \leq \kappa$). The servers are distinct in all these executions (thus there are $q(\kappa + \binom{\kappa}{2})$ servers in all), but the input and output clients in these different executions are identified as follows. There are 2κ input

clients I_{j0} and I_{j1} with inputs x_{j0} and x_{j1} respectively for $j = 1, \dots, \kappa$; η_j has (I_{j0}, I_{j1}) as its two input clients; $\phi_{jj'}$ has $(I_{j0}, I_{j1}, I_{j'0}, I_{j'1})$ as its four clients. There are $Lm + 1$ clients $Z_{\ell 0}^i$ and $Z_{\ell 1}^i$, (for $i = 1, \dots, m$ and $\ell = 1, \dots, L$) and Z^0 , which serve as the output clients in *all* κ instances of η . Further Z^0 will serve as the output client in all the $\binom{\kappa}{2}$ instances of ϕ .

In these executions the inputs (x_{j0}, x_{j1}) are set independently for each j as a random additive sharing of the input to cOT^{OT} , Γ ; i.e., $x_{j0} \oplus x_{j1} = \Gamma$ for each j .

- *Cut and choose:* Using a single round of multiple (1-out-of- N) OTs, the sender and the receiver do the following:
 - For each j , the sender sends views of the pair of input clients (I_{j0}, I_{j1}) via a 1-out-of-2 OT channel, and the receiver picks one view at random.
 - In each of the κ executions of η and each of the $\binom{\kappa}{2}$ executions of ϕ , the sender makes *two* lists of the q server views, and sends each list via a 1-out-of- q OT channel. From each list, the receiver selects independently at random one server's view.
 - For each $i = 1, \dots, m$, the sender sends the views of $(Z_{\ell 0}^i, Z_{\ell 1}^i)$ for $\ell = 1, \dots, L$ through L 1-out-of-2 OT channels. The receiver picks the views $Z_{\ell r_\ell}^i$ for a random $r \in \{0, 1\}^L$ such that $\bigoplus_{\ell} r_\ell = c_i$.

In addition, the sender sends the view of the Z^0 directly (i.e., without using OT).

- The receiver checks for consistency in the input it received:
 1. *The input clients:* Views of all the exposed input clients are locally correct, i.e., each input client's view is according to its program given its initial input and random tapes (In particular each of them feeds the same input to the instance of η as well as to the $\kappa - 1$ instances of ϕ that it participates in.)
 2. *The servers and the "edges":*
 - The views of the exposed servers are locally correct (given the incoming messages and the random tapes).
 - The views of the exposed servers are consistent with the incoming messages reported in the views of the exposed output clients and the outgoing messages implicit in the views of the exposed input clients.
 - The views of the exposed servers are consistent with each other (in particular, if the same server was exposed twice, the two views are identical).
 3. *The output clients:*
 - In all executions $\phi_{jj'}$, ($1 \leq j < j' \leq \kappa$), Z^0 outputs 1. Also, in all executions η_j , ($1 \leq j \leq \kappa$), Z^0 produces the same output (say γ).
 - The views of (i.e., outputs produced by) all the exposed output clients (including Z^0) are correct given the incoming messages.
 - Let $z_{j\ell b}^i$ denote the output of the output client $Z_{\ell b}^i$ in the η_j . Then, for each i , $\bigoplus z_{j\ell r_\ell}^i$ evaluates to the same value (say \tilde{s}^i) for all j .

If all the verifications succeed, then the receiver outputs $(\tilde{s}^1, \dots, \tilde{s}^m; \gamma)$. Else it aborts the protocol by sending *abort* to the sender.

Lemma 1 *Given a protocol η satisfying the conditions above, cOT^{OT} defined above is a UC-secure realization of the certified OT functionality $\mathcal{F}_{\text{cov-cOT}}$. The simulation is perfect.*

PROOF OVERVIEW: The interesting cases are when exactly one of the sender or the receiver is corrupted.

Corrupt Receiver. In this case, the security easily follows from the privacy of the protocol η . Consider a simulator in the ideal world interacting with $\mathcal{F}_{\text{cov-cOT}}$ as the receiver, and simulating the protocol to the corrupt receiver. Note that the only things a receiver can do in the protocol is to select which server views it obtains, and which input client's view it gets to see. First the simulator extracts (c_1, \dots, c_m) from observing which output clients' views the receiver requests via the OT functionality (using default values if necessary). Then it sends these to $\mathcal{F}_{\text{cov-cOT}}$ and obtains the outputs for the receiver. Observe that the views that the receiver can obtain in each execution of η or ϕ are of up to 2 servers, input clients (with only one share of an additive sharing of Γ) and some output clients. By the security guarantee on η and ϕ , this view can be perfectly simulated given just the outputs that these output clients receive. Since these outputs are available to the simulator, it can carry out a perfect simulation.

Corrupt Sender. This case is the more interesting one.

Consider the graph on the parties in the protocol with an edge between two parties who can exchange a message in the protocol. (That is, there are edges between the input clients and the servers, among the q servers, and between the servers and the output clients.)

Let δ be the minimum probability of detecting an “internally” inconsistent execution of ϕ or of η . Note that $\delta \geq 1/q^2$.

The simulator obtains the entire collection of views that the sender submits as inputs to the OT executions. It examines these views and first prepares an “input consistency graph” as follows: For each j ($j = 1, \dots, \kappa$), such that both I_{j0} and I_{j1} are locally consistent, add a node to the graph. For each pair (j, j') such that the entire execution of $\phi_{jj'}$ is correct (given the randomness of the servers), add an edge between the corresponding nodes (if present) in the graph. Note that for any connected component in this graph, there is a unique input value Γ such that for all j in the connected component, $x_{j0} \oplus x_{j1} = \Gamma$ and the input clients of η_j use this input.

Now the simulator proceeds as follows:

- The simulator sends the `cheat` command to $\mathcal{F}_{\text{cov-cOT}}$ if any of the following conditions hold.
 1. The input consistency graph has no connected component of more than $\kappa/2$ nodes.
 2. η_j was internally consistent only for $\kappa/2$ or fewer values of j .
 3. For some i , for all ℓ ($\ell = 1, \dots, L$), output client $Z_{\ell 0}^i$ or $Z_{\ell 1}^i$ was locally incorrect.

Then, with probability ϵ , $\mathcal{F}_{\text{cov-cOT}}$ will send `corrupted` to both parties; in this case the simulator aborts the simulated protocol. With probability $1 - \epsilon$, $\mathcal{F}_{\text{cov-cOT}}$ will allow the simulator to cheat: the simulator obtains the inputs of the receiver and calculates the probability p that the simulator would have aborted the protocol. We shall see that $p > \epsilon$. Then with probability $(p - \epsilon)/(1 - \epsilon)$ the simulator will abort the simulated protocol (so that total probability of the simulated protocol being aborted is exactly p) and send `corrupted` to $\mathcal{F}_{\text{cov-cOT}}$; with probability $(1 - p)/(1 - \epsilon)$ it will continue the simulation conditioned on the receiver not aborting, derive the output that the receiver obtains, and send this to $\mathcal{F}_{\text{cov-cOT}}$ as the output for the receiver.

- If the above conditions do not hold (and so the simulator does not send `cheat` to $\mathcal{F}_{\text{cov-cOT}}$), then
 1. The simulator can derive an input Γ , which is the input defined by the majority of the nodes in the input consistency graph.
 2. Also, since more than $\kappa/2$ executions of η were internally consistent, there is some j such that η_j was internally consistent and used inputs x_{j0} and x_{j1} such that $x_{j0} \oplus x_{j1} = \Gamma$.
 3. Further, for each i , for at least one ℓ (ℓ can depend on i), both $Z_{\ell 0}^i$ and $Z_{\ell 1}^i$ were locally correct (for all κ execution of η).

In this case the simulator proceeds to give a perfect simulation as follows. Note that the only information the sender learns is whether the receiver aborts the protocol or not. The simulator carries out all the checks like the receiver, except for the last step. For the last check, the receiver would pick, for each i , $r \in \{0, 1\}^L$ such that $\bigoplus_{\ell} r_{\ell} = c_i$. But the simulator (who does not know c_i), simply picks a random string r . However, this is equivalent to picking r' where the ℓ -th bit of r is flipped. This is because both the views $Z_{\ell 0}^i$ and $Z_{\ell 1}^i$ are locally correct. Thus the simulated protocol is a perfect simulation of the real protocol so far.

If the simulated protocol does not abort, then the simulator sends Γ to $\mathcal{F}_{\text{cov-cOT}}$. Otherwise it sends `corrupted` to $\mathcal{F}_{\text{cov-cOT}}$. By the correctness of η_j for some j , we know that the output of the receiver in the real protocol is perfectly simulated by the output $\mathcal{F}_{\text{cov-cOT}}$ delivers in the ideal world, on input Γ .

To complete the argument we need to argue that the probability p of the real protocol aborting in the three cases where the simulator would send `cheat` to $\mathcal{F}_{\text{cov-cOT}}$ is indeed at least ϵ . We consider the three cases below.

1. If the input consistency graph has no more than $\kappa/2$ nodes in a single connected component, then there must be either $\Omega(\kappa)$ missing nodes (i.e., j for which the node was not added to the graph), or $\Omega(\kappa^2)$ missing edges (i.e., edges (j, j') that were not added to the graph).
 - For each missing node j , one of the two input clients I_{j0} and I_{j1} is locally incorrect (sending different inputs to executions of η and ϕ). If there are d such missing nodes, the probability of the real protocol aborting is at least $1 - 2^{-d}$, because for each j there is an independent probability of at least half of exposing an incorrect view.
 - For each missing edge (j, j') (between nodes which are present in the input consistency graph), the probability of the protocol aborting is δ if $\phi_{jj'}$ is internally inconsistent, or is 1, if Z^0 is either locally incorrect or produces an output 0 in $\phi_{jj'}$. If there are d such missing edges, the probability of the protocol aborting is at least $1 - (1 - \delta)^{-d}$ (as these events are independent of each other).

In any of these cases, the real protocol execution would abort with probability at least $p_1 = 1 - (1 - \delta)^{\Omega(\kappa)}$.

2. If η_j was internally consistent only for $\kappa/2$ or fewer values of j , then the protocol will abort with probability at least $p_2 = 1 - (1 - \delta)^{\kappa/2}$.
3. If for some i , for all ℓ ($\ell = 1, \dots, L$), at least one of the output clients $Z_{\ell 0}^i$ and $Z_{\ell 1}^i$ was locally incorrect, then the protocol would abort with probability at least $p_3 = 1 - 2^{-(L-1)}$.

To ensure that these abort probabilities are at least ϵ , we set $\epsilon := \min(p_1, p_2, p_3)$. Note that with $L = 2$, and a large enough κ , we can get $\epsilon = \frac{1}{2}$. By choosing $L = \Omega(\kappa)$, we get $\epsilon = 1 - 2^{-\Omega(\kappa)}$. \triangleleft