Output Compression, MPC, and iO for Turing Machines

Saikrishna Badrinarayanan[1(✉)], Rex Fernando[1], Venkata Koppula[2], Amit Sahai[1], and Brent Waters[3]

[1] UCLA, Los Angeles, USA
{saikrishna,rex,sahai}@cs.ucla.edu
[2] Weizmann Institute of Science, Rehovot, Israel
venkata.koppula@weizmann.ac.il
[3] UT Austin and NTT Research, Austin, USA
bwaters@cs.utexas.edu

Abstract. In this work, we study the fascinating notion of output-compressing randomized encodings for Turing Machines, in a *shared randomness model*. In this model, the encoder and decoder have access to a shared random string, and the efficiency requirement is, the size of the encoding must be independent of the running time and output length of the Turing Machine on the given input, while the length of the shared random string is allowed to grow with the length of the output. We show how to construct output-compressing randomized encodings for Turing machines in the shared randomness model, assuming iO for circuits and any assumption in the set {LWE, DDH, Nth Residuosity}.

We then show interesting implications of the above result to basic feasibility questions in the areas of secure multiparty computation (MPC) and indistinguishability obfuscation (iO):

1. **Compact MPC for Turing Machines in the Random Oracle Model.** In the context of MPC, we consider the following basic feasibility question: does there exist a malicious-secure MPC protocol for Turing Machines whose communication complexity is independent of the running time and output length of the Turing Machine when executed on the combined inputs of all parties? We call such a protocol as a *compact* MPC protocol. Hubácek and Wichs [HW15] showed via an incompressibility argument, that, even for the restricted setting of circuits, it is impossible to construct a malicious secure two party computation protocol in the plain model where the communication complexity is independent of the output length. In this work, we show how to evade this impossibility by compiling any (non-compact) MPC protocol in the plain model to a *compact* MPC protocol for Turing Machines in the Random Oracle Model, assuming output-compressing randomized encodings in the shared randomness model.
2. **Succinct iO for Turing Machines in the Shared Randomness Model.** In all existing constructions of iO for Turing Machines, the size of the obfuscated program grows with a bound on the input length. In this work, we show how to construct an iO scheme for

© International Association for Cryptologic Research 2019
S. D. Galbraith and S. Moriai (Eds.): ASIACRYPT 2019, LNCS 11921, pp. 342–370, 2019.
https://doi.org/10.1007/978-3-030-34578-5_13

Turing Machines in the shared randomness model where the size of the obfuscated program is independent of a bound on the input length, assuming iO for circuits and any assumption in the set {LWE, DDH, Nth Residuosity}.

1 Introduction

In this work, we study the fascinating notion of output-compressing randomized encodings for Turing machines. We explore the implication of such encodings to a natural and surprisingly unexplored form of secure multiparty computation for Turing Machines, and also to indistinguishability obfuscation for Turing Machines.

Output-compressing randomized encodings were introduced in the works of Ananth and Jain [AJ15] and Lin, Pass, Seth and Telang [LPST16] as a generalization of randomized encodings [IK00] and succinct randomized encodings [KLW15,BGL+15,CHJV15]. Recall that in an output-compressing randomized encoding scheme for Turing machines, there exists an encode algorithm that takes as input a Turing machine M and an input x. It outputs an encoding $\widetilde{M_x}$ such that the decode algorithm, given this encoding $\widetilde{M_x}$, can compute the output $M(x)$. The efficiency requirement is that for any machine M and input x, the size of the encoding is $\mathsf{poly}(|M|, |x|, \lambda)$, for some fixed polynomial poly, where λ is the security parameter. In particular, the size of the encoding should be *independent of the output length* and the running time of the machine M on input x.[1] In those papers, the authors defined both indistinguishability based and simulation based security notions. In this work, we will focus on the stronger notion of simulation based security. This notion requires an output-compressing randomized encoding scheme to have a corresponding simulator Sim, that, for any Turing machine M and input x, given just the output $M(x)$, along with the size of the machine $|M|$ and the input length $|x|$, outputs a simulated encoding $\widetilde{M_x}$ that is indistinguishable from a real encoding of the machine M and input x.[2] As stated here, this goal is impossible in the standard model due to an "incompressibility" argument as shown by Lin et al. [LPST16]. Such incompressibility arguments have been a source of impossibility proofs in many areas of cryptography such as functional encryption, garbled circuits and secure multiparty computation [BSW11,AIKW13,CIJ+13,AGVW13,HW15] and this is perhaps the reason why simulation secure output compressing randomized encodings have not been well-studied so far.

Our starting observation is that the above impossibility fails to hold in a *shared randomness model*, where the size of the randomness can grow with

[1] The size can depend logarithmically on the output length and running time.

[2] We actually consider a stronger notion where part of the input need not be hidden, and we require that the size of the encoding should not grow with this revealed part. This is a generalization of the notion of partial garbling schemes introduced by Ishai and Wee [IW14].

the output length. More formally, both the encoder and decoder share a random string (whose size can grow with the output length) and we require two properties: (1) For any machine M and input x, the size of the encoding is $\mathsf{poly}(|M|, |x|, \lambda)$, for some fixed polynomial poly. (2) There exists a simulator Sim, that, for any Turing machine M and input x, given just the output $M(x)$, along with the length of the machine $|M|$ and the input length $|x|$, outputs a pair of a simulated encoding $\widetilde{M_x}$ and a shared random string that is indistinguishable from the pair of a real encoding and a uniformly random string.

Our first main result is that we can, in fact, construct output-compressing randomized encodings for Turing machines in the shared randomness model, assuming indistinguishability obfuscation (iO) for circuits along with any assumption in {Decisional Diffie Hellman (DDH), Learning With Errors (LWE), N^{th} Residuosity} where the size of the shared randomness equals the output length. Recall that iO is necessary because output-compressing randomized encodings for Turing machines implies iO for circuits as shown by Lin et al. [LPST16] (it is easy to see that this implication to iO remains true even in the shared randomness model). We describe the techniques used in our construction in Sect. 2.1. We then use this new tool to tackle basic feasibility questions in the context of two fundamental areas in Cryptography: secure multiparty computation (MPC) and indistinguishability obfuscation (iO).

Compact MPC for Turing Machines with Unbounded Output in the Random Oracle model. The first basic feasibility question we address is the following: Consider a set of n mutually distrusting parties with inputs $x_1, \ldots, x_n$ respectively that agree on a Turing machine M. Their goal is to securely compute the output $M(x_1, \ldots, x_n)$ without leaking any information about their respective inputs, where we stress that the output can be of any unbounded polynomial size. Crucially, we require that the communication complexity of the protocol (the sum of the length of the messages exchanged by all the parties) is $\mathsf{poly}(|M|, |x_1|, \ldots, |x_n|, \lambda)$ for some fixed polynomial poly where λ is the security parameter. In particular, the communication complexity should be independent of the output length and the running time of the machine M on input $(x_1, \ldots, x_n)$. We call such an MPC protocol to be *compact*. Indeed, this communication efficiency requirement is the most natural efficiency requirement in the context of MPC for Turing machines.

Remarkably, this extremely basic question, in the context of Turing machines, has never been considered before to the best of our knowledge (see related work below for comparison with previous work). At first glance, one may think that Fully Homomorphic Encryption (FHE), one of the most powerful primitives in Cryptography, should help solve this problem. The reason being that, at least in the two party setting, FHE allows one party to encrypt its input and send it to the other party, who can then homomorphically evaluate the function to be computed "under the hood" and compute an encryption of the final output. However, its then not clear how this evaluator would learn the output since he does not have the decryption key. Sending the encryption of the final output to the other party would also blow up the communication complexity. This is related to the question posed by Hubácek and Wichs [HW15], where they consider a circuit based model, a special case of our notion. That is, they consider n parties who wish to securely evaluate a circuit on their joint inputs such that the communication complexity of the protocol is independent of the output length of the circuit. They show how to achieve semi-honest secure two party computation with this efficiency requirement assuming iO for circuits and a somewhere statistically binding (SSB) hash function. Further, they showed that in the context of malicious adversaries,[3] in the standard model, it is impossible to construct a secure computation protocol with such efficiency requirement even for just two party computation.

Our approach to this problem is motivated by an unwillingness to give up on malicious secure compact MPC. To that end we must find a way to evade the impossibility result, and we do so by considering the well-studied programmable random oracle (RO) model [BR93,Nie02,DSW08,Wee09,CJS14,CDG+18]. We stress that the RO model is typically exploited in pursuit of efficiency improvements, but here we are seeking to use it to establish basic feasibility results. Indeed, the RO model has enabled important feasibility results in the past which were impossible in the plain model, for example unconditional non-interactive zero-knowledge arguments for NP with sub-linear communication [IMS12] and Universal Samplers [HJK+16]. In addition, a straightforward modification of the impossibility argument in [HW15] shows the programmable RO model is the weakest model in which we can hope to obtain simulation-secure compact MPC. In the programmable RO model, the simulator is allowed to choose the RO's responses to the adversary *adaptively*, based on the adversary's previous messages. If we restrict the simulator to choosing the RO's responses *selectively*, before any interaction with the adversary, then this model is not sufficient to achieve compact MPC. This also rules out the possibility of compact MPC in the CRS model.[4] Aside from its theoretical interest in terms of basic feasibility, our work motivates the following question: is there a weaker notion of security for MPC for which compact MPC is realizable in the plain model? A full answer to this question is outside the scope of this paper, but we believe that this is an excellent topic for future work. As a starting point, in the full version of our paper we sketch a simple example where the techniques in our paper can still yield meaningful security guarantees in the plain model.

More specifically, we show how to construct a *compact* constant round MPC protocol for Turing machines in the RO model secure against malicious adversaries, assuming iO for circuits and any assumption in {DDH, LWE, Nth Residuosity}. Recall that by *compact*, we mean that the communication complexity of the protocol is independent of the output length and running time of the Turing machine being evaluated on the joint inputs of the parties. We obtain this result by using output-compressing randomized encodings in the shared randomness

[3] Their impossibility in fact even ruled out the simpler setting of honest but deterministic adversaries - such an adversary behaves honestly in the protocol execution but fixes its random tape to some deterministic value.

[4] See the full version of the paper for a full presentation of this argument, based on Theorem 4.3 in [HW15].

model to compile any non-compact malicious secure constant round MPC protocol (even just for circuits) in the plain model into a *compact* constant round MPC protocol for Turing machines in the RO model while preserving the round complexity. We again stress that to the best of our knowledge, this is the first MPC protocol for Turing machines where the communication complexity is bounded by a polynomial in the description length of the machine and the input lengths of all the parties. We also observe that as a corollary of our work, we obtain the first malicious secure compact MPC protocol in the circuit based model of Hubácek and Wichs [HW15], in the RO model. We describe the techniques used in our construction in Sect. 2.2.

Succinct iO for Turing Machines for Bounded Inputs in the Shared Randomness Model. The problem of bootstrapping from iO for circuits to iO for Turing machines has been the subject of intense study over the last few years. In 2015, in three concurrent works [KLW15, BGL+15, CHJV15][5] showed how to construct iO for Turing machines where the size of the obfuscation grows with a bound on the input length to the Turing machine. In this work, we ask the following question: can we construct iO for Turing machines in the shared randomness model where the obfuscator and evaluator have a shared random string that grows with the input bound but the size of the obfuscation does not?

Lin et al. [LPST16] showed that output-compressing randomized encodings are closely related to iO for Turing machines. That is, they showed that simulation secure output-compressing randomized encodings in the plain model implies iO for Turing machines with unbounded inputs.[6] In particular, this implies iO for Turing machines with bounded inputs where the size of the obfuscation does not grow with the input bound. As we know, simulation secure output-compressing randomized encodings are impossible in the plain model. However, in turns out that this implication does not carry over in the shared randomness model. That is, if we start with output-compressing randomized encodings in the shared randomness model and apply the transformation in [LPST16], in the resulting iO scheme, the size of the obfuscation does in fact grow with the input bound. The key obstacle is that in the transformation, the obfuscation consists of an output-compressing randomized encoding that is the root of a GGM-like tree [GGM86]. This encoding, on evaluation, outputs another output-compressing randomized encoding corresponding to its child node and the process is repeated. In order to evaluate the obfuscated program on an input of length n, the evaluator has to traverse the obfuscated program up to a depth of length n. As a result, the machine being encoded in the root needs the shared randomness for each layer, up to a depth of length n. Hence, the size of the machine encoded in the root

[5] Recently, concurrent to our work, [AL18, AM18, GS18] also showed how to construct iO for Turing machines where, similar to [KLW15, BGL+15, CHJV15], the size of the obfuscation grows with a bound on the input length to the Turing machine.

[6] Lin et al. [LPST16] in fact showed that a weaker notion of distributional indistinguishability based secure output-compressing randomized encodings suffices to imply iO for Turing machines with unbounded inputs. However, they also supplement this by showing that it is impossible, in general, to construct such encodings.

grows with the input bound and so does the size of the obfuscated program. Note that this approach fails even if the size of the shared randomness for the encoding is just 1 bit (independent of the length of the output).

We show how to overcome this obstacle by taking a completely different approach. In our solution, the obfuscated program consists of an output-compressing randomized encoding in which, crucially, neither the machine being encoded nor the input to the machine, depends on the input bound of the obfuscation scheme. Hence, the size of the encoding, and therefore, also the size of the obfuscation, does not grow with the input bound. We elaborate more about the techniques used in our construction in Sect. 2.3. Concretely, letting n denote the input bound, we obtain iO for Turing machines M in the shared randomness model where the size of the obfuscation is $\mathsf{poly}_1(|M|, \lambda)$ for some fixed polynomial poly_1, and where the obfuscator and evaluator have a shared random string of length $\mathsf{poly}(n, \lambda)$ for some fixed polynomial poly. Our assumptions are again iO for circuits and any assumption in {DDH, LWE, Nth Residuosity}.

On Reuse of the Shared Randomness. We note that it is possible for multiple output-compressing randomized encodings to reuse the shared randomness in a limited way. Namely, if we have several output-compressing randomized encodings, and we can construct hybrids such that only one randomized encoding needs to be simulated at a time, then all of the encodings can share the same CRS. This applies to the succinct iO construction: multiple circuits can be obfuscated using a single shared random string. Moreover, modulo a preprocessing phase which can be shared among all obfuscations, the time to obfuscate M is independent of the output length or running time of M.

1.1 Our Results

In this paper, we achieve the following results.

(1) Output-compressing randomized encodings.
We prove the following theorem:

Theorem 1 (Informal). *There exists an output-compressing randomized encoding scheme for Turing machines in the shared randomness model assuming the existence of:*

- *iO for circuits (AND)*
- *$A \in$ {DDH, LWE, Nth Residuosity}.*

Further, the length of the shared randomness is equal to the output length.

(2) Compact MPC for Turing machines with unbounded output in the RO model.
We prove the following theorem:

Theorem 2 (Informal). *For any n, $t > 0$, there exists a constant round* compact *MPC protocol amongst n parties for Turing machines in the Programmable Random Oracle model that is malicious secure against up to t corruptions assuming the existence of:*

- *Output-compressing randomized encodings in the shared randomness model (AND)*
- *Constant round MPC protocol amongst n parties in the plain model that is malicious secure against up to t corruptions.*

Once again, recall that by *compact*, we mean that the communication complexity of the protocol is independent of the output length and running time of the Turing machine being evaluated on the joint inputs of the parties. Here, we note that the above compiler even works if the underlying MPC protocol is for circuits. That is, we can convert any constant round protocol for circuits into a constant round protocol for Turing machines (with an input bound) by first converting the Turing machine into a (potentially large) circuit.

Also, we can instantiate the underlying MPC protocol in the following manner to get a round optimal compact MPC: append a non-interactive zero knowledge argument based on DLIN in the common random string model [GOS06] to either the two round semi-malicious MPC protocol of [MW16] that is based on LWE in the common random string model or the ones of [GS18,BL18] that are based on DDH/N^{th} residuosity in the plain model, to get two round malicious secure MPC protocols in the common random string model. We can then implement the common random string required for the underlying protocol via the RO. We thus achieve the following corollary:

Corollary 1. *Assuming the existence of:*

- *iO for circuits (AND)*
- *DDH, or LWE, or N^{th} Residuosity (AND)*
- *DLIN,*

there exists a compact, round optimal *(two round) MPC protocol π for Turing machines in the Programmable Random Oracle model that is malicious secure against a dishonest majority.*

Our result also gives a malicious secure *compact* MPC protocol in the circuit-based setting of [HW15] in the RO model. We also achieve other interesting corollaries by instantiating the underlying MPC protocol in the setting of super-polynomial simulation or in the setting of concurrent executions. We elaborate on both the above points in Sect. 6.

(3) Succinct iO for Turing machines for bounded inputs in the shared randomness model.
We prove the following theorem:

Theorem 3 (Informal). *There exists an iO scheme for Turing machines in the shared randomness model where the size of the obfuscated program is independent of the input bound assuming the existence of:*

- *iO for circuits,*
- *DDH, or LWE, or N^{th} Residuosity.*

1.2 Related Work

Lin, Pass, Seth and Telang [LPST16] construct OCREs from compact functional encryption (which is implied by iO), in the common *reference* string model. This is different from the shared randomness model in that the CRS which is shared among all parties must be generated in a specific fashion: in particular, [LPST16] require that the CRS be a specific function secret key. This model requires more trust be placed in the trusted setup phase. We note that our construction of compact MPC requires strong OCREs in the shared randomness model; [LPST16] does not consider strong OCREs, but even if a construction did exist in the common reference string model, to the best of our knowledge, it would not be sufficient to construct compact MPC.

A series of works [OS97,GHL+14,GGMP16,Mia16,HY16,LO17] consider MPC for RAM programs. However, in all of them, the communication complexity of the protocol grows with the running time of the RAM program. As a result, the communication complexity of the protocol in the Turing machine model would also grow with the output length. We stress that in our work, we require that the communication complexity can grow with neither output length nor running time of the Turing machine.

Ananth et al. [AJS17] construct an iO scheme for Turing machines in which, for any machine M and input bound L, the size of the obfuscation is $|M| + \mathsf{poly}(L, \lambda)$. However, in our setting, we require that the size be independent of this bound L.

2 Technical Overview

2.1 Output Compressing Randomized Encodings

We will now discuss a high-level overview of our output-compressing randomized encoding (OcRE) scheme in the *shared randomness model*. Let $\mathcal{M}$ be a family of Turing machines with output size bounded by $\mathsf{o\text{-}len}$. An OcRE scheme for $\mathcal{M}$ in the shared randomness model consists of a setup algorithm, an encoding algorithm and a decoding algorithm. The setup algorithm takes as input security parameter λ together with a string rnd of length $\mathsf{o\text{-}len}$, and outputs a succinct encoding key ek of size $\mathsf{poly}(\lambda)$.[7] This encoding key is used by the encoding algorithm, which takes as input a machine $M \in \mathcal{M}$, an input $x \in \{0,1\}^*$, and outputs an encoding $\widetilde{M_x}$. Finally, the decoding algorithm can use $\widetilde{M_x}$ and rnd to recover $M(x)$. For efficiency, we require that the encoding time depends only on $|M|$, $|x|$ and security parameter λ. In particular, the size of the encoding should not grow with the output length $\mathsf{o\text{-}len}$ or the running time of M on x.[8]

[7] We will assume $\mathsf{o\text{-}len}$ is at most 2^λ.

[8] Strictly speaking, it is allowed to depend polylogarithmically on the running time of M on input x; for this overview, we will ignore this polylogarithmic dependence on the running time.

The starting point of our construction is the succinct randomized encoding scheme of [KLW15], which is an encoding scheme for *boolean* Turing machines, and the size of the encoding depends only on $|M|, |x|$ and security parameter λ. We want to use this tool as a building block to build an encoding scheme for *general* Turing machines (i.e. with multi-bit output) where the size of the encoding still only depends on $|M|, |x|$ and λ. As a first step, let us consider the following approach. The encoding algorithm outputs an obfuscated program $\mathsf{Prog}[M, x]$, which has M and x hardwired, takes input $j \in [\mathsf{o\text{-}len}]$, and outputs a KLW encoding of M_j, x (the randomness for computing the encoding is obtained by applying a PRF on j). Here, M_j is a boolean Turing machine which, on input x, outputs the j^{th} bit of $M(x)$. The decoding algorithm runs Prog for each $j \in [\mathsf{o\text{-}len}]$, obtains $\mathsf{o\text{-}len}$ different encodings, and then decodes each of them to obtain the entire output bit by bit. Clearly, this construction satisfies the efficiency requirement. This is because the size of the program Prog depends only on $|M|, |x|$, and hence the size of the encoding only depends on $|M|, |x|, \lambda$. As far as security is concerned, it is easy to show that this scheme satisfies indistinguishability-based security; that is, if (M_0, x_0) and (M_1, x_1) are two pairs such that $M_0(x_0) = M_1(x_1)$, $|M_0| = |M_1|$, $|x_0| = |x_1|$, then the obfuscation of $\mathsf{Prog}[M_0, x_0]$ is computationally indistinguishable from the obfuscation of $\mathsf{Prog}[M_1, x_1]$. Unfortunately, recall that our goal is simulation security, and it is not possible to simulate an obfuscation of $\mathsf{Prog}[M, x]$, given only $M(x)$ as input. In particular, if $y = M(x)$ is a long pseudorandom string (whose length can be much longer than the size of $\mathsf{Prog}[M,]$), then should be hard to compress y to a short encoding (as shown by Lin et al. [LPST16]).

As noted in the previous section, we will evade the "incompressibility" argument by allowing the shared randomness to have size that grows with the output length. Our goal will be to allow the simulator to embed the output of the machine M in this randomness. Our second attempt is as follows. The setup algorithm computes a short commitment ek to the shared randomness (say with a Merkle tree), and outputs ek as the encoding key. The encoding algorithm computes an obfuscation of $\mathsf{Prog}[M, x, \mathsf{ek}]$, which has M, x, ek hardwired, takes as input an index j, a bit b (which is supposed to be the j^{th} bit of the shared randomness), and an opening π that the bit b is indeed the j^{th} bit of the shared random string. The program checks the proof π, and then computes a KLW encoding of (M_j, x).

While the bit b is essentially ignored in the real-world encoding, it is used by the simulator in the ideal world. In the ideal world, the simulator, on receiving $M(x)$, masks it with a pseudorandomly generated one-time pad and outputs the resultant string as the shared randomness, and the short commitment ek is computed as in real world. For the encoding, it outputs an obfuscation of $\mathsf{Prog\text{-}sim}[\mathsf{ek}]$, which takes as input (j, b, π), checks the proof π, unmasks the bit b to obtain $M(x)_j$ and simulates the KLW randomized encoding using $M(x)_j$. This program has behavior identical to $\mathsf{Prog}[M, x, \mathsf{ek}]$ as long as the adversary only gives openings to the original bits of the shared randomness.

There is a simple problem with this idea: obfuscation only guarantees indistinguishability of programs that are functionally equivalent, and although the security of a Merkle tree would make it computationally infeasible for an adversary to come up with an opening to a wrong value, these inputs do in fact exist. To fix this problem, we use a special iO-compatible family of hash functions called 'somewhere-statistically binding (SSB) hash', introduced by [HW15]. Intuitively, this primitive is similar to a merkle tree except for two additional features. First, it allows a given position to be statistically "bound", where for that index it is only possible to give an opening for the correct bit. So there are three algorithms, $\mathsf{Setup}, \mathsf{Open}$, and Verify, as in the case of a Merkle tree, but Setup additionally takes as input a position to bind. If j is the bound position for H then there is no opening π for a bit $b \neq x_j$ such that $\mathsf{Verify}(\pi, b, j, H(x))$ accepts. Second, this bound position is hidden, so we can change it without being detected. Using this new hash allows us to make a series of hybrids where we change the shared randomness one bit at a time without giving up indistinguishability.

2.2 Compact MPC for Turing Machines in the Random Oracle Model

We now describe the techniques used in our round preserving compiler from any non-compact constant round malicious secure MPC protocol in the plain model to a *compact* constant round malicious secure MPC protocol in the RO model, using output-compressing randomized encodings in the shared randomness model.

To begin with, consider any constant round MPC protocol π in the plain model. For simplicity, lets assume that every party broadcasts a message in each round. In order to make it compact, our main idea is a very simple one: use output-compressing randomized encodings to compress the messages sent by every party in each round so that they are independent of the output length and running time of the machine. That is, instead of sending the actual message of protocol π, each party just sends an output-compressing randomized encoding of a machine and its private input that generates the actual message.

More precisely, consider a party P with input x that intends to send a message msg_1 in the first round as part of executing protocol π. Let's denote M to be the Turing machine that all the parties wish to evaluate. Let M_1 denote the algorithm used by the first party to generate this message msg_1 in the first round. Now, instead of sending msg_1, P sends an encoding of machine M_1 and input (x, r) where r is the randomness used by party P in protocol π. The recipient first decodes this encoding to receive P's first round message of protocol π - msg_1. Without loss of generality, let's assume that the length of randomness r is only proportional to the input length (else, internally, M_1 can apply a pseudorandom generator). In terms of efficiency, the description of the machine M_1 only depends on M, and so it is easy to see that the size of the encoding does not depend on the non-compact message msg_1. A natural initial observation is that in order to construct a simulator for the protocol, we need to generate simulated encodings. However, as we know that simulation secure output-compressing randomized encodings are impossible, we will resort to using our new encodings constructed in the shared randomness model.

Need for Random Oracle. Recall that in the introduction we ruled out the possibility of malicious-secure compact MPC in the CRS model. As a result, it must be the case that our protocol is not a compact and secure MPC protocol in the CRS model. We illustrate what goes wrong with a naive use of our output-compressing randomized encodings. After receiving a message in the first round from every other party, P first decodes all these messages to compute a transcript trans for protocol π. P then computes an encoding of machine M_2 and input (x, r, trans) where M_2 is the machine used to generate the next message msg_2 and sends this in round 2. Looking ahead to the security proof, the simulator will have to generate a simulated encoding of this message and also simulate the shared randomness. To do that, the simulated shared randomness will have to depend on $M_2(x, r, \mathsf{trans})$. Notice that the simulator will have to decide the simulated CRS *before* beginning the protocol execution. This is not possible, however, because the value trans depends on the adversary's input and randomness, both of which are not even picked before the adversary receives the CRS.

We use the programmable RO model to circumvent this. Now, in each round, along with its encoding, P also sends a short index. The recipient first queries the RO on this index to compute the shared randomness that is then used to decode. Looking ahead to the proof, the simulator can pick a random index that the RO has not been queried on so far and "program" the RO's output to be the simulated shared randomness. This can be executed after receiving the transcript of the previous round and before sending the pair of index and simulated encoding in any round.

Strong Output-Compressing Randomized Encodings. Next, it turns out that, in fact, just standard output-compressing randomized encodings do not suffice for the above transformation. To see why, consider any round j. Let trans denote the transcript of the underlying protocol π at the end of round $(j-1)$. Now, in round j, party P sends an encoding of machine M_j and input (x, r, trans), where M_j is the machine used to generate the j^{th} round message. However, the size of trans could depend on the output length of the protocol because trans denotes the transcript of the underlying non-compact protocol π. A natural attempt to solve would be to let trans be the transcript of the new compact protocol up to this point instead of the underlying protocol, and to let M_j decode the transcript when forming the next message. This also turns out to be problematic, though, since we now need a randomized encoding of a machine M_j which accesses the RO. As a result, since the size of the encoding in each round grows with the input to the machine being encoded, the size of the messages in each round also does depend on the output length. Thus we are seemingly back to square one, since our transformation still yields a non-compact protocol.

In order to solve this issue, we make the crucial observation that the part of the input to the machine being encoded that actually grows with the output length of the protocol is actually public information. That is, we do not care about any privacy for this part of the input and only require that the size of the encoding does not grow with this public input. Corresponding to this, we define a new stronger version of output-compressing randomized encodings in the shared randomness model, which we call *strong* output-compressing randomized encodings. In more detail, the encoding algorithm takes as input a machine M, a private input x_1 and a public input x_2 and outputs an encoding. Informally, the efficiency requirement is that the size of the encoding is $\mathsf{poly}(|M|, |x_1|)$ for a fixed polynomial poly and does not depend on x_2, in addition to being independent of the output length and running time. Further, security requires that, in addition to the output $M(x_1, x_2)$, the simulator is also given the public input x_2 and the tuple of honest encoding and honest shared randomness should be indistinguishable from the tuple of simulated encoding and simulated shared randomness. Thus, if we use strong output-compressing randomized encodings, we overcome the issue. Our construction of strong output-compressing randomized encodings is very similar to the construction in Sect. 2.1 except that we replace the succinct notion called succinct partial randomized encodings. More details can be found in Sect. 5.

Another subtle detail is that, while proving security, in the sequence of hybrids, it is essential that we first switch the encodings to be simulated before switching the messages of the protocol π from real to simulated. This is because we can not afford to send honest encodings of simulated messages of protocol π as the description of the simulator's machine to generate these messages could grow with the output bound. One interesting consequence of the above point is that our transformation is oblivious to whether the underlying simulator rewinds or runs in super-polynomial time. As a result, our construction naturally extends even to the setting of concurrent security if the underlying protocol is concurrently secure.

Notice that our compiler to solve this very basic feasibility question is in fact, remarkably simple, which further highlights the power of simulation secure output-compressing randomized encodings in the shared randomness model. We refer the reader to Sect. 6 for more details about our compact MPC protocol and proof.

Implication in the Circuit Model of [HW15]. First, recall that in the setting of Hubácek and Wichs [HW15], the goal is to construct an MPC protocol for circuits where the communication complexity is independent of the output length of the circuit. At first glance, it might seem that our construction trivially implies a result in the circuit setting as well. However, this is not quite directly true. Observe that in our protocol, the communication complexity grows with the description of the Turing machine and so, when we convert the circuit to the Turing machine model, the communication complexity grows with the size of the circuit. In the case of a circuit, the output length can in fact be proportional to the size of the circuit. To circumvent this, we will consider a Turing machine representation of a Universal circuit, that takes as input a circuit C and an input x and evaluates $C(x)$. Now, notice that the size of this universal circuit, and by extension, the size of the Turing machine evaluated, is independent of the circuit being evaluated. Further, we will set the circuit being computed - C, to be part of the "public" input to each strong output-compressing randomized encoding that is computed in each round of the protocol. Since all parties have knowledge of C, we don't need to hide this input. As a result, neither the machine being encoded nor the private input depend on the circuit being evaluated and this solves the problem. That is, the communication complexity of the resulting compiled protocol is independent of the output length of the circuit.

2.3 Succinct iO for Turing Machines in the Shared Randomness Model

We now describe the techniques used in our construction of iO for Turing machines in the shared randomness model where the size of the obfuscated program does not grow with a bound on the input length. We will denote such obfuscation schemes as *succinct* iO schemes in this section. First, we recall from the introduction that the transformation of Lin et al. [LPST16] to go from output-compressing randomized encodings to succinct iO does not work in the shared randomness model. Briefly, the reason was that if we want to support Turing machines with input length n, then there must be n chunks of the shared randomness, and the 'top-level' encoding in the LPST scheme must contain a commitment to each of the n chunks, and as a result, the size grows with n.

Therefore, our obfuscation scheme will have a completely different structure. Recall that [KLW15] showed an obfuscation scheme where the size of obfuscation of M with input bound n grows with the security parameter, input bound and machine size (but does not depend on the running time of M on any input). We will use such *weakly-succinct* obfuscation scheme to obtain succinct $i\mathcal{O}$.

Consider a program P that takes as input a Turing machine M, input bound n, and outputs a weakly-succinct obfuscation of M with input bound n (the randomness for obfuscation can be generated using a pseudorandom generator). The size of the output grows with n, size of M and security parameter λ. But the important thing to note here is that the size of program P does not grow with input bound n. Therefore, we can use output-compressing randomized encodings to construct succinct iO. The obfuscation algorithm simply outputs an encoding of program P with inputs (M, n). Clearly, the size of this encoding does not grow with n (using the efficiency property of ocre). The proof of security follows from the security of the obfuscation scheme and the output-compressing randomized encoding scheme.

Finally, an informed reader might recall that the LPST construction required the security parameter to grow at each level, while in our case, we can work with a single security parameter. The reason for this is because their security reduction loses a factor of 2 for each level, and therefore the security parameter must grow at each level. In our case, we have a different proof structure, and the switch from encoding of P, M_0 to P, M_1 in the security proof is a single-step jump.

Organization. We first describe some preliminaries in Sects. 3 and 4. In Sect. 5, we describe the construction of strong output-compressing randomized encodings for Turing machines in the shared randomness model. Then, in Sect. 6, we construct compact MPC protocols in the random oracle model. Finally, we defer to the full version of the paper our construction of succinct iO for Turing machines and succinct partial randomized encodings.

3 Preliminaries

We will use λ to denote the security parameter throughout the rest of the paper. For any string s of length n, let $s[i]$ denote the i^{th} bit of s. Without loss of generality, we assume all Turing machines are oblivious.

We defer to the full version of our paper for some additional preliminaries, including the definition of secure multiparty computation in the random oracle model.

4 Randomized Encodings: Definitions

4.1 Succinct Partial Randomized Encodings

In this section, we introduce the notion of succinct partial randomized encodings (spRE). This is similar to the notion of succinct randomized encodings, except that the adversary is allowed to learn part of the input. For efficiency, we require that if the machine has size m, and ℓ bits of input are hidden, then the size of randomized encoding should be polynomial in the security parameter λ, ℓ and m. In particular, the size of the encoding does not depend on the entire input's length (this is possible only because we want to hide ℓ bits of the input; the adversary can learn the remaining bits of the input). This notion is the Turing Machine analogue of *partial garbling* of arithmetic branching programs, studied by Ishai and Wee [IW14].

A succinct partial randomized encoding scheme SPRE for a class of boolean Turing machines $\mathcal{M}$ consists of a preprocessing algorithm $\mathsf{Preprocess}$, encoding algorithm Encode, and a decoding algorithm Decode with the following syntax.

$\mathsf{Preprocess}(1^\lambda, x_2 \in \{0,1\}^*)$: The preprocessing algorithm takes as input security parameter λ (in unary), string $y \in \{0,1\}^*$ and outputs a string hk.

$\mathsf{Encode}(M \in \mathcal{M}, T \in \mathbb{N}, x_1 \in \{0,1\}^*, \mathsf{hk} \in \{0,1\}^{p(\lambda)})$: The encoding algorithm takes as input a Turing machine $M \in \mathcal{M}$, time bound $T \in \mathbb{N}$, partial input $x_1 \in \{0,1\}^*$, string $\mathsf{hk} \in \{0,1\}^{p(\lambda)}$, and outputs an encoding $\widetilde{M}$.

$\mathsf{Decode}(\widetilde{M}, x_2, \mathsf{hk})$: The decoding algorithm takes as input an encoding $\widetilde{M}$, a string $x_2 \in \{0,1\}^*$, string hk and outputs $y \in \{0,1,\perp\}$.

Definition 1. *Let $\mathcal{M}$ be a family of Turing machines. A randomized encoding scheme* $\mathsf{SPRE} = (\mathsf{Preprocess}, \mathsf{Encode}, \mathsf{Decode})$ *is said to be a succinct partial randomized encoding scheme if it satisfies the following correctness, efficiency and security properties.*

– *Correctness: For every machine $M \in \mathcal{M}$, string $x = (x_1, x_2) \in \{0,1\}^*$, security parameter λ and $T \in \mathbb{N}$, if* $\mathsf{hk} \leftarrow \mathsf{Preprocess}(1^\lambda, x_2)$*, then* $\mathsf{Decode}(\mathsf{Encode}(M, T, x_1, \mathsf{hk}), x_2) = \mathsf{TM}(M, x, T)$*.*
– *Efficiency: There exist polynomials $p_{\mathsf{prep}}, p_{\mathrm{enc}}$ and p_{dec} such that for every machine $M \in \mathcal{M}$, $x = (x_1, x_2) \in \{0,1\}^*$, $T \in \mathbb{N}$ and $\lambda \in \mathbb{N}$, if* $\mathsf{hk} \leftarrow \mathsf{Preprocess}(1^\lambda, x_2)$*, then $|\mathsf{hk}| = p_{\mathsf{prep}}(\lambda)$, the time to encode $\widetilde{M} \leftarrow$* $\mathsf{Encode}(M, T, x_1, \mathsf{hk})$ *is bounded by $p_{\mathrm{enc}}(|M|, |x_1|, \log T, \lambda)$, and the time to decode $\widetilde{M}$ is bounded by* $\min(\mathsf{Time}(M, x, T) \cdot p_{\mathrm{dec}}(\lambda, \log T)$*.*
– *Security: For every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a PPT simulator $\mathcal{S}$ such that for all PPT distinguishers $\mathcal{D}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$,* $\Pr[1 \leftarrow \mathcal{D}(\mathsf{Expt\text{-}SPRE\text{-}Real}_{\mathsf{SPRE},\mathcal{A}}(\lambda))] - \Pr[1 \leftarrow \mathcal{D}(\mathsf{Expt\text{-}SPRE\text{-}Ideal}_{\mathsf{SRE},\mathcal{A},\mathcal{S}}(\lambda))] \leq$ *negl(λ), where* Expt-SPRE-Real *and* Expt-SPRE-Ideal *are defined in Fig. 1. Moreover, the running time of $\mathcal{S}$ is bounded by some polynomial $p_{\mathcal{S}}(|M|, |x_1|, \log T, \lambda)$.*

Experiments $\mathsf{Expt\text{-}SPRE\text{-}Real}_{\mathsf{SPRE},\mathcal{A}}(\lambda)$ **and** $\mathsf{Expt\text{-}SPRE\text{-}Ideal}_{\mathsf{SPRE},\mathcal{A},\mathcal{S}}(\lambda)$

$\mathsf{Expt\text{-}SPRE\text{-}Real}_{\mathsf{SPRE},\mathcal{A}}(\lambda)$:

- $(M, x = (x_1, x_2), T, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$.
- $\mathsf{hk} \leftarrow \mathsf{Preprocess}(x_2, 1^\lambda)$.
- $\widetilde{M} \leftarrow \mathsf{Encode}(M, T, x_1, \mathsf{hk})$.
- Experiment outputs $\mathcal{A}_2(\widetilde{M}, \sigma)$.

$\mathsf{Expt\text{-}SPRE\text{-}Ideal}_{\mathsf{SPRE},\mathcal{A},\mathcal{S}}(\lambda)$:

- $(M, x = (x_1, x_2), T, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$, $t^* = \min(T, \mathsf{Time}(M, x))$, out $= \mathsf{TM}(M, x, T)$.
- $\mathsf{hk} \leftarrow \mathsf{Preprocess}(1^\lambda, x_2)$.
- $\widetilde{M} \leftarrow \mathcal{S}\left(1^{|M|}, 1^{|x_1|}, \mathsf{hk}, 1^\lambda, \mathrm{out}, t^*\right)$.
- Experiment outputs $\mathcal{A}_2(\widetilde{M}, \sigma)$.

Fig. 1. Simulation Security Experiments for partial randomized encodings

Our construction of succinct partial randomized encodings is closely related to the succinct randomized encodings scheme by [KLW15] and we defer the details to the full version of our paper.

4.2 Strong Output-Compressing Randomized Encodings in the Shared Randomess Model

The notion of succinct randomized encodings (defined in the full version of our paper) was originally defined for boolean Turing machines. We can also consider randomized encodings for Turing machines with long outputs. Using (standard) succinct randomized encodings, one can construct randomized encodings for Turing machines with multi-bit outputs, where the size of encodings grows linearly with the output size. In a recent work, Lin et al. [LPST16] introduced a stronger notion called *output-compressing randomized encodings*, where the size of the encoding only depends sublinearly on the output length. Lin et al. also showed that simulation based security notions of output-compressing randomized encodings are impossible to achieve. In this work, we consider a stronger notion of output-compressing randomized encodings in the shared randomness model where the encoder and decoder have access to a shared random string (denoted by crs). Here, the machine also takes another public input x_2 along with a private input x_1 with the requirement that the size of the encoding should only grow polynomially in the size of the machine and the private input x_1. In particular, it does not grow with x_2 or the running time of the machine or its output length. We define it formally below.

A strong output-compressing randomized encoding scheme $\mathsf{S.OcRE} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ in the shared randomness model consists of three algorithms with the following syntax.

$\mathsf{Setup}(1^\lambda, 1^{\mathsf{o\text{-}len}}, \mathsf{crs} \in \{0,1\}^{\mathsf{o\text{-}len}})$: The setup algorithm takes as input security parameter λ, output-bound $\mathsf{o\text{-}len}$ and a shared random string crs of length $\mathsf{o\text{-}len}$. It outputs an encoding key ek.

$\mathsf{Encode}((M, \mathsf{tmf}(\cdot)), x = (x_1, x_2), T, \mathsf{ek})$: The encoding algorithm takes as input an oblivious Turing Machine M with tape movement function $\mathsf{tmf}(\cdot)$, input x consisting of a private part x_1 and a public part x_2, time bound $T \leq 2^\lambda$ (in binary) and an encoding key ek, and outputs an encoding $\widetilde{M_x}$.

$\mathsf{Decode}(\widetilde{M_x}, x_2, \mathsf{crs})$: The decoding algorithm takes as input an encoding $\widetilde{M_x}$, a public input x_2, the shared random string crs and outputs $y \in \{0,1\}^* \cup \{\perp\}$.

Definition 2. *A strong output-compressing randomized encoding scheme* $\mathsf{S.OcRE} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ *in the shared randomness model is said to be secure if it satisfies the following correctness, efficiency and security requirements.*

– *Correctness: For all security parameters $\lambda \in \mathbb{N}$, output-length bound* $\mathsf{o\text{-}len} \in \mathbb{N}$, $\mathsf{crs} \in \{0,1\}^{\mathsf{o\text{-}len}}$, *machine M with tape movement function* $\mathsf{tmf}(\cdot)$, *input $x = (x_1, x_2)$, time bound T such that* $|M(x)| \leq \mathsf{o\text{-}len}$, *if* $\mathsf{ek} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{o\text{-}len}}, \mathsf{crs})$, $\widetilde{M_x} \leftarrow \mathsf{Encode}((M, \mathsf{tmf}(\cdot)), x, T\mathsf{ek})$, *then* $\mathsf{Decode}(\widetilde{M_x}, x_2, \mathsf{crs}) = \mathsf{TM}(M, x, T)$.
– *Efficiency: There exist polynomials p_1, p_2, p_3 such that for all $\lambda \in \mathbb{N}$,* $\mathsf{o\text{-}len} \in \mathbb{N}$, $\mathsf{crs} \in \{0,1\}^{\mathsf{o\text{-}len}}$*:*
  1. *If* $\mathsf{ek} \leftarrow \mathsf{Setup}(1^\lambda, 1^o, \mathsf{crs})$, $|\mathsf{ek}| \leq p_1(\lambda, \log o)$.
  2. *For every Turing machine M, time bound T, input $x = (x_1, x_2) \in \{0,1\}^*$, if* $\widetilde{M_x} \leftarrow \mathsf{Encode}(M, x, T, \mathsf{ek})$, *then* $|\widetilde{M_x}| \leq p_2(|M|, |x_1|, \log|x_2|, \log T, \log o, \lambda)$.
  3. *The running time of* $\mathsf{Decode}(\widetilde{M_x}, x_2, \mathsf{crs})$ *is at most* $\min(T, \mathsf{Time}(M, x)) \cdot p_3(\lambda, \log T)$.
– *Security: For every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a simulator $\mathcal{S}$ such that for all PPT distinguishers $\mathcal{D}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\Pr[1 \leftarrow \mathcal{D}(\mathsf{Expt\text{-}S.OcRE\text{-}Real}_{\mathsf{S.OcRE},\mathcal{A}}(\lambda))] - \Pr[1 \leftarrow \mathcal{D}(\mathsf{Expt\text{-}S.OcRE\text{-}Ideal}_{\mathsf{S.OcRE},\mathcal{A},\mathcal{S}}(\lambda))] \leq negl(\lambda),$$

where $\mathsf{Expt\text{-}S.OcRE\text{-}Real}$ *and* $\mathsf{Expt\text{-}S.OcRE\text{-}Ideal}$ *are defined in Fig. 2.*

Experiments $\mathsf{Expt\text{-}S.OcRE\text{-}Real}_{\mathsf{S.OcRE},\mathcal{A}}(\lambda)$ **and** $\mathsf{Expt\text{-}S.OcRE\text{-}Ideal}_{\mathsf{S.OcRE},\mathcal{A},\mathcal{S}}(\lambda)$

$\mathsf{Expt\text{-}S.OcRE\text{-}Real}_{\mathsf{S.OcRE},\mathcal{A}}(\lambda)$:

- $(1^{\mathsf{o\text{-}len}}, (M, \mathsf{tmf}(\cdot)), x = (x_1, x_2), T, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$.
- $\mathsf{crs} \leftarrow \{0,1\}^{\mathsf{o\text{-}len}}$, $\mathsf{ek} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{o\text{-}len}}, \mathsf{crs})$.
- $\widetilde{M} \leftarrow \mathsf{Encode}((M, \mathsf{tmf}(\cdot)), x, T, \mathsf{ek})$.
- Experiment outputs $\mathcal{A}_2(\mathsf{crs}, \mathsf{ek}, \widetilde{M}, \sigma)$.

$\mathsf{Expt\text{-}S.OcRE\text{-}Ideal}_{\mathsf{S.OcRE},\mathcal{A}}(\lambda)$:

- $(1^{\mathsf{o\text{-}len}}, (M, \mathsf{tmf}(\cdot)), x = (x_1, x_2), T, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$.
- Let $t^* = \min(T, \mathsf{Time}(M, x))$ and $b^* = \mathsf{TM}(M, x, T)$.
- $s \leftarrow \mathcal{S}(1^{|M|}, 1^{|x_1|}, \mathsf{tmf}(\cdot), x_2, t^*, b^*, 1^\lambda)$.
- Let $s = (\mathsf{crs}, \widetilde{M})$.
- $\mathsf{ek} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{o\text{-}len}}, \mathsf{crs})$.
- Experiment outputs $\mathcal{A}_2(\mathsf{crs}, \mathsf{ek}, \widetilde{M}, \sigma)$.

Fig. 2. Simulation Security Experiments for strong output-compressing randomized encodings in the shared randomness model

Remark: In particular, note that strong output-compressing randomized encodings ($\mathsf{S.OcRE}$) implies output-compressing randomized encodings (OcRE) by setting the public input x_2 to be $\perp$.

5 Strong Output-Compressing Randomized Encodings in the CRS Model

In this section, we show a construction of strong output-compressing randomized encodings in the common random string (CRS) model. Formally, we show the following theorem:

Theorem 4. *Assuming the existence of iO for circuits* and *somewhere statistically binding (SSB) hash* and *Puncturable PRFs* and *Succinct partial randomized encodings for single-bit output Turing machines, There exists a strong output-compressing randomized encoding scheme for Turing machines in the shared randomness model.*

Instantiating the SSB hash and the succinct partial randomized encodings, we get the following corollary:

Corollary 2. *Assuming the existence of iO for circuits* and *any $A \in \{$DDH, LWE, N^{th} Residuosity$\}$, There exists a strong output-compressing randomized encoding scheme for Turing machines in the shared randomness model.*

Notation and Primitives Used: We will be using the following cryptographic primitives for our construction:

– Indistinguishability obfuscation for circuits ($\mathsf{Ckt.Obf}$, $\mathsf{Ckt.Eval}$).
– Succinct partial randomized encodings for single-bit output Turing machines ($\mathsf{SPRE.Preprocess}$, $\mathsf{SPRE.Encode}$, $\mathsf{SPRE.Decode}$). Without loss of generality, we assume that the algorithm $\mathsf{SPRE.Encode}$ uses λ bits of randomness - it can internally apply a PRG on this randomness if a larger amount is required.

– Somewhere statistically binding hash($\mathsf{SSB.Gen}$, $\mathsf{SSB.Open}$, $\mathsf{SSB.Verify}$).
– A Puncturable PRF (F_1, $\mathsf{PPRF.Puncture}_1$) that takes inputs of size λ and outputs 1 bit.
– A Puncturable PRF (F_2, $\mathsf{PPRF.Puncture}_2$) that takes inputs of size λ and outputs λ bits.

5.1 Construction

$\mathsf{S.OcRE.Setup}(1^\lambda, 1^o, \mathsf{crs} \in \{0,1\}^o)$: The setup algorithm does the following:
1. Choose hash function $H \leftarrow \mathsf{SSB.Gen}(1^\lambda, o, 0)$.[9]
2. Compute $h = H(\mathsf{crs})$ and set $\mathsf{ek} = (h, H)$.

$\mathsf{S.OcRE.Encode}(M, x = (x_1, x_2), T, \mathsf{ek} = (h, H))$: The encoding algorithm does the following:
1. Compute $\mathsf{hk} = \mathsf{SPRE.Preprocess}(1^\lambda, x_2)$.
2. Choose a key K_{SPRE} for the puncturable PRF F_2.
3. Let M_i denote the turing machine that, on input x, runs the machine M on input x and outputs the i^{th} bit of $M(x)$. Let t denote $|\mathsf{SPRE.Encode}(M_i, T, x_1, \mathsf{hk}; r)|$ using any random string r.
4. Compute $\widetilde{\mathsf{Prog}} \leftarrow \mathsf{Ckt.Obf}(\mathsf{Prog}, 1^\lambda)$ where the program Prog is defined in Fig. 3. Note that the size of the program Prog is padded appropriately so that it is equal to the size of the program $\mathsf{Prog\text{-}sim}$ defined later in Fig. 4.
5. Output $\widetilde{M_x} = (\widetilde{\mathsf{Prog}}, t, H)$.

$\mathsf{S.OcRE.Decode}(\widetilde{M_x} = (\widetilde{\mathsf{Prog}}, t, H), x_2, \mathsf{crs})$: For each $i \in [o]$, the decoding algorithm computes bit out_i as follows:

Prog

Inputs : Index $i \in [o]$, bit $\mathsf{str} \in \{0,1\}$, proof π, index $j \in [t]$.
Hardwired : Hash value $h \in \{0,1\}^\lambda$, machine M, input x_1, PRF key K_{SPRE}, bound T, preprocessed value hk.

1. Verify proof π : Check if $\mathsf{SSB.Verify}(H, h, i, \mathsf{str}, \pi) = 1$. If not, output $\perp$.
2. Recall that M_i denotes the turing machine that, on input x, runs the machine M on input x and outputs the i^{th} bit of $M(x)$. Compute out $=$ $\mathsf{SPRE.Encode}(M_i, T, x_1, \mathsf{hk}; F_2(K_{\mathsf{SPRE}}, i))$ and output the j^{th} bit of out.

Fig. 3. Circuit Prog

[9] We modify the syntax of the SSB hash system slightly to allow the binding index to range from $0, \ldots, o$ and without loss of generality, just set $\mathsf{SSB.Gen}(1^\lambda, o, 0) = \mathsf{SSB.Gen}(1^\lambda, o, 1)$. That is, when the binding index is set as 0, we actually don't care at what index the hash system is bound at and will not actually use the statistically binding property. This is just to be consistent with the definition of the SSB hash system.

1. Parse $\mathsf{crs} = (\mathsf{crs}[1], \mathsf{crs}[2], \ldots, \mathsf{crs}[o])$, where each $\mathsf{crs}[j]$ is a bit.
2. Compute SSB proof for each $\mathsf{crs}[j]$; that is, compute $\pi[j] = \mathsf{SSB.Open}(H, \mathsf{crs}, j)$.
3. For $j = 1$ to t, do:
   (a) Compute $\widetilde{M_i}[j] = \mathsf{Ckt.Eval}(\widetilde{\mathsf{Prog}}, (i, \mathsf{crs}[i], \pi[i], j))$.
4. Let $\widetilde{M_i} = (\widetilde{M_i}[1]\ \widetilde{M_i}[2]\ \ldots\ \widetilde{M_i}[t])$. Compute $\mathrm{out}_i = \mathsf{SPRE.Decode}(\widetilde{M_i}, x_2)$.

Finally, it outputs $(\mathrm{out}_1\ \mathrm{out}_2\ \ldots\ \mathrm{out}_o)$.

Correctness and Succinctness. Correctness follows from the correctness of (SPRE.Encode,SPRE.Decode) and (Ckt.Obf, Ckt.Eval).

Below we show the three efficiency properties required by the definition.

1. If $\mathsf{ek} \leftarrow \mathsf{Setup}(1^\lambda, 1^o, \mathsf{crs})$, $|\mathsf{ek}| = \ell_{\mathrm{hash}}(\lambda) + \ell_{\mathrm{fn}}(\lambda)$, where ℓ_{hash} and ℓ_{fn} are from SSB.
2. For every Turing machine M, time bound T, input $x = (x_1, x_2) \in \{0,1\}^*$, if $\widetilde{M_x} \leftarrow \mathsf{Encode}(M, x, T, \mathsf{ek})$, then $|\widetilde{M_x}| = (|prog| + |t|) \leq |\mathsf{Prog}| + \mathsf{poly}(\lambda)$. Prog is padded to be the same length as the programs used in the hybrids and Prog-sim, so $|\mathsf{Prog}|$ is the maximum of the length of these programs. By inspecting the values hardwired in each of these programs we get $|\mathsf{Prog}| \leq p(|h|, |M|, |x_1|, |hk|, k, \log o, t)$, where k is the maximum size of the keys of F_1 and F_2. By the efficiency of SPRE, the definition of SSB hashes and the definition of puncturable PRFs we get that $|\mathsf{Prog}| \leq p_2(\lambda, |M|, |x_1|, \log o)$ and thus $|\widetilde{M_x}| \leq p_2(\lambda, |M|, |x_1|, \log |x_2|, \log o)$ for some fixed polynomial p_2.
3. The running time of $\mathsf{Decode}(\widetilde{M_x}, x_2, \mathsf{crs})$ is at most $O(o \times t_1 + o \times t \times t_2)$ where t_1 is the running time of $\mathsf{SPRE.Decode}(\widetilde{M_i}, x_2)$ and t_2 is the running time of $\mathsf{Ckt.Eval}(\widetilde{\mathsf{Prog}}, (i, \mathsf{crs}[i], \pi[i], j))$. By the efficiency of the SPRE scheme and the iO scheme we have $\mathsf{Decode}(\widetilde{M_x}, x_2, \mathsf{crs}) \leq \min(T, \mathsf{Time}(M, x)) \cdot p_3(\lambda, \log T)$.

5.2 Proof of Security

Description of Simulator. The simulator S.OcRE.Sim gets as input the value $M(x)$ (which is the output of the machine M on input x) and the public part of the input x_2, and it must simulate the shared random string crs and an encoding $\widetilde{M_x}$ of the machine M and x. We now describe the simulator.

S.OcRE.Sim$(\mathbf{1^{|M|}}, \mathbf{1^{|x_1|}}, \mathbf{x_2}, \mathbf{1^\lambda}, \mathbf{M(x)}, \mathbf{T})$:
The simulator does the following:

1. Compute $\mathsf{hk} = \mathsf{SPRE.Preprocess}(1^\lambda, x_2)$.
2. Choose a key K_{crs} for the puncturable PRF F_1 and a key K_{sim} for the puncturable PRF F_2.
3. Then, for each i, compute $\mathsf{crs}[i] = M(x)_i \oplus w^i$ where $w^i = F(K_{\mathsf{crs}}, i)$ and $M(x)_i$ denotes the i^{th} bit of $M(x)$. The shared random string is set to be $(\mathsf{crs}[1]\ \mathsf{crs}[2]\ \ldots\ \mathsf{crs}[o])$.
4. Choose a hash function $H \leftarrow \mathsf{SSB.Gen}(1^\lambda, o, 0)$ and compute $h = H(\mathsf{crs})$.

5. Compute $\widetilde{\mathsf{Prog\text{-}sim}} \leftarrow \mathsf{Ckt.Obf}(\mathsf{Prog\text{-}sim}, 1^\lambda)$, where $\mathsf{Prog\text{-}sim}$ is defined in Fig. 4.
6. Let M_i denote the turing machine that, on input x, runs the machine M on input x and outputs the i^{th} bit of $M(x)$. Let t denote $|\mathsf{SPRE.Encode}(M_i, T, z, \mathsf{hk}; r)|)$ using any random string r and any input z such that $|z| = |x_1|$.
7. Set $\widetilde{M_x} = (\widetilde{\mathsf{Prog\text{-}sim}}, t)$.

```
                                   Prog-sim

Inputs : Index $i \in [o]$, bit $\mathsf{str} \in \{0,1\}$, proof $\pi$, index $j \in [t]$
Hardwired : Hash $h \in \{0,1\}^\lambda$, machine $M$, PRF keys $K_{\mathrm{sim}}, K_{\mathsf{crs}}$, preprocessed value $\mathsf{hk}$.

  1. Verify proof $\pi$ : Check if $\mathsf{SSB.Verify}(H, h, i, \mathsf{str}, \pi) = 1$. If not, output $\perp$.
  2. Do the following:
     (a) Let $w = F_1(K_{\mathsf{crs}}, i)$ and $y = w \oplus \mathsf{str}$.
     (b) Compute out $= \mathsf{SPRE.Sim}(1^{|M_i|}, 1^{|x_1|}, \mathsf{hk}, 1^\lambda, y, T; r)$ where $r = F_2(K_{\mathrm{sim}}, i)$.
     (c) Output $j^{th}$ bit of out.
```

Fig. 4. Simulated program $\mathsf{Prog\text{-}sim}$

Hybrids. We will show that the real and ideal worlds are indistinguishable via a sequence of $(o+2)$ hybrid experiments Hyb_0 to Hyb_{o+1} where Hyb_0 corresponds to the real world and Hyb_{o+1} corresponds to the ideal world. For each $i \in [o]$, in hybrid Hyb_{i^*}, the first i^* bits of the CRS are computed as encryptions of output bits (with the w's as one time pads). The encoding of M, x does not compute the SRE for $i \le i^*$. More formally:

Hybrid Hyb_{i^*}:
The challenger does the following:

1. Compute $\mathsf{hk} = \mathsf{SPRE.Preprocess}(1^\lambda, x_2)$.
2. Choose a key K_{crs} for the puncturable PRF F_1 and two keys $K_{\mathrm{sim}}, K_{\mathsf{SPRE}}$ for the puncturable PRF F_2.
3. Then, for each $i \le i^*$, compute $\mathsf{crs}[i] = M(x)_i \oplus w^i$ where $w^i = F_1(K_{\mathsf{crs}}, i)$ and $M(x)_i$ denotes the i^{th} bit of $M(x)$.
4. For each $i > i^*$, pick $\mathsf{crs}[i]$ uniformly at random.
5. The shared random string is set to be $(\mathsf{crs}[1]\ \mathsf{crs}[2]\ \ldots\ \mathsf{crs}[o])$.
6. Choose a hash function $H \leftarrow \mathsf{SSB.Gen}(1^\lambda, o, i^*)$ and compute $h = H(\mathsf{crs})$. Set $\mathsf{ek} = h$.
7. Compute $\widetilde{\mathsf{Prog}\text{-}i^*} \leftarrow \mathsf{Ckt.Obf}(\mathsf{Prog}\text{-}i^*, 1^\lambda)$, where $\mathsf{Prog}\text{-}i^*$ is defined in Fig. 5.
8. Let M_i denote the turing machine that, on input x, runs the machine M on input x and outputs the i^{th} bit of $M(x)$. Let t denote $|\mathsf{SPRE.Encode}(M_i, T, x_1, \mathsf{hk}; r)|$ using any random string r.
9. Set $\widetilde{M_x} = (\widetilde{\mathsf{Prog}\text{-}i^*}, t)$.

```
Prog-$i^*$

Inputs : Index $i \in [o]$, bit $\mathsf{str} \in \{0,1\}$, proof $\pi$, index $j \in [t]$
Hardwired : Index $i^*$, Hash $h \in \{0,1\}^\lambda$, machine $M$, input $x_1$, PRF keys
$K_{\mathsf{crs}}, K_{\mathrm{sim}}, K_{\mathsf{SPRE}}$, bound $T$, preprocessed value $\mathsf{hk}$.

1. Verify proof $\pi$ : Check if $\mathsf{SSB.Verify}(H, h, i, \mathsf{str}, \pi) = 1$. If not, output $\perp$.
2. If $i \leq i^*$, do the following:
    (a) Let $w = F_1(K_{\mathsf{crs}}, i)$ and $y = w \oplus \mathsf{str}$.
    (b) Compute out $=$ $\mathsf{SPRE.Sim}(1^{|M_i|}, 1^{|x_1|}, \mathsf{hk}, 1^\lambda, y, T; r)$ where $r =$
        $F_2(K_{\mathrm{sim}}, i)$.
    (c) Output $j^{th}$ bit of out.
3. Else, if $i > i^*$: Recall that $M_i$ denotes the turing machine that, on input $x$,
   runs the machine $M$ on input $x$ and outputs the $i^{th}$ bit of $M(x)$. Compute
   out $= \mathsf{SPRE.Encode}(M_i, T, x_1, \mathsf{hk}; F_2(K_{\mathsf{SPRE}}, i))$ and output the $j^{th}$ bit of
   out.
```

Fig. 5. Hybrid program Prog-i^*

Hybrid Hyb_{o+1}:
Identical to Hyb_o except that the value x_1 is not hardwired into Prog-i^*.

We include the proof of hybrid indistinguishability in the full version of the paper.

6 Compact MPC

We consider the problem of constructing a malicious secure compact MPC protocol for Turing machines. Consider a set of n mutually distrusting parties with inputs $x_1, \ldots, x_n$ respectively that agree on a TM M. Their goal is to securely compute the output $M(x_1, \ldots, x_n)$ without leaking any information about their respective inputs where the output can be of any unbounded polynomial size. We first define the notion of a *compact* MPC protocol. Let λ denote the security parameter and let $\mathsf{Comm.Compl}(\pi)$ denote the communication complexity (sum of the lengths of all messages exchanged by all parties) of any protocol π. Let $\mathsf{Time}(\mathcal{M}, \mathsf{x})$ denote the running time of turing machine $\mathcal{M}$ on input x.

Definition 3. *An MPC protocol π is said to be* compact *if there exists a fixed polynomial* poly *such that for all machines M and inputs $(x_1, \ldots, x_n)$,* $\mathsf{Comm.Compl}(\pi) = \mathsf{poly}(|M|, |x_1|, \ldots, |x_n|, \lambda, \log(\mathsf{Time}(\mathcal{M}, \mathsf{x})))$. *In particular, the communication complexity is independent of the output length and the running time of the machine on the inputs of all the parties.*

In this section, we give a round preserving compiler from any constant round (non-compact) malicious secure MPC protocol in the plain model to a malicious secure *compact* MPC protocol for Turing machines in the random oracle (RO) model.

Formally, we prove the following theorem:

Theorem 5. *For all $n, t > 0$, assuming the existence of:*

- *A (constant) k round*[10] *MPC protocol amongst n parties in the plain model that is malicious secure against up to t corruptions (AND)*
- *Strong OCRE in the shared randomness model,*

there exists a k round compact *MPC protocol π amongst n parties for Turing machines in the Programmable Random Oracle model that is malicious secure against up to t corruptions.*

Here, we note that the above compiler even works if the underlying MPC protocol is for circuits. That is, we can convert any constant round protocol for circuits into a constant round protocol for Turing machines (with an input bound) by first converting the Turing machine into a (potentially large) circuit.

Corollaries:
We can instantiate the strong OCRE from our construction in Sect. 5. We now discuss several corollaries on instantiating the underlying MPC protocol with various protocols in literature based on different models.

1. Instantiating the MPC protocol with the round optimal[11] plain model construction of [BGJ+18] that is secure against a dishonest majority based on DDH/Nth Residuosity, we get a four round *compact* MPC protocol π for Turing machines in the RO model that is secure against a dishonest majority assuming iO for circuits and DDH/Nth Residuosity.
2. We can also instantiate the underlying MPC protocol with protocols that are secure in the Common Random String model by using the RO's output on some fixed string to implement the common random string. In particular, combining the two round semi-malicious MPC protocol of [MW16] that is based on LWE in the common random string model or the ones of [GS18, BL18] that are based on DDH/N^{th} residuosity in the plain model, with a non-interactive zero knowledge argument based on DLIN in the common random string model [GOS06], we get two round malicious secure MPC protocols in the common random string model. As a result, we have the following corollary:

Corollary 3. *Assuming the existence of iO for circuits* and *A where $A \in \{LWE, DDH, N^{th}$ Residuosity$\}$* and *DLIN, there exists a round optimal (two round)* compact *MPC protocol π for Turing machines in the Programmable Random Oracle model that is malicious secure against a dishonest majority.*

3. We note that our transformation works even on instantiating the underlying constant round MPC protocol with ones that are secure in the setting of super-polynomial simulation [Pas03, BGI+17] or in the concurrent (self-composable) setting [GGJS12, BGJ+17] to yield *compact* versions of the same in the RO model.

[10] Observe that our round preserving compiler in fact works for any MPC protocol where the number of rounds is independent of the machine being evaluated.

[11] Recall that in the plain model, the optimal round complexity is 4.

Implication to [HW15] Model. Finally, we observe that our transformation also has an implication to the circuit-based model of Hubácek and Wichs [HW15] as elaborated in Sect. 2.2. Thus, we get the following corollary:

Theorem 6. *For all $n, t > 0$, assuming the existence of a constant round MPC protocol amongst n parties in the plain model that is malicious secure against up to t corruptions,* and *strong OCRE in the shared randomness model, there exists a constant round MPC protocol π amongst n parties for all polynomial sized circuits in the RO model that is malicious secure against up to t corruptions where the communication complexity of the protocol is independent of the output length of the circuit. That is, there exists a fixed polynomial* poly *such that, for all circuits* C *and all inputs* $(x_1, \ldots, x_n) \in \mathsf{Domain}(\mathsf{C})$*,* $\mathsf{Comm.Compl}(\pi) = \mathsf{poly}(|x_1|, \ldots, |x_n|, \lambda)$*.*

6.1 Construction

Notation and Primitives Used:

- Let λ denote the security parameter and RO be a random oracle that takes as input a tuple $(r, 1^{\mathsf{len}})$ where $|r| = \lambda$ and outputs a string of length len.
- Consider n parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ with inputs $\mathsf{x}_1, \ldots, \mathsf{x}_n$ respectively (with $|\mathsf{x}_i| = \lambda$ for each $i \in [n]$) who wish to evaluate any turing machine $\mathcal{M}$ on their joint inputs.
- Let $\mathsf{S.OcRE} = (\mathsf{S.OcRE.Setup}, \mathsf{S.OcRE.Encode}, \mathsf{S.OcRE.Decode})$ be a strong OCRE scheme in the shared randomness model.
- Let π^{plain} be a t round MPC protocol for turing machines in the plain model that is malicious secure against a dishonest majority. For simplicity, we assume that the protocol works using a broadcast channel - that is, in each round, every party broadcasts a message to all other parties.
- Let $\mathsf{NextMsg}_k(\cdot)$ denote the algorithm used by any party to compute the k^{th} round of protocol π^{plain} and let $\mathsf{Out}(\cdot)$ denote the algorithm used by any party to compute the final output. Also, without loss of generality, assume that in protocol π^{plain}, each party uses randomness rand_i of length λ.[12]

Remark: To ease the exposition, we assume that the Random Oracle can output arbitrarily long strings by also taking the desired output length as input to the oracle. In reality, let's say it outputs strings of length $p(\lambda)$ where p is a polynomial. Then, in the protocol below, each party can output a starting query index $r_{i,j}$ and an offset $o_{i,j}$ to indicate that the shared random string is actually the concatenation of $\mathsf{RO}(r_{i,j}), \ldots, \mathsf{RO}(r_{i,j} + o_{i,j})$. Note that $|o_{i,j}| \leq \lambda$.

[12] Internally, we can apply a PRG to expand this to any length of randomness we require. Here, we are implicitly assuming that the protocol requires each party to use uniformly random strings. This is true of almost every constant round MPC protocol.

Protocol: The protocol is described below.

1. **Round 1:**
   Each party P_i does the following:
   - Pick a random string $r_{i,1} \in \{0,1\}^\lambda$. Let $\mathsf{len}_{i,1} = |\mathsf{NextMsg}_1(\mathsf{x}_i; \mathrm{rand}_i)|$.
   - Compute $\mathsf{crs}_{i,1} = \mathsf{RO}(r_{i,1}, 1^{\mathsf{len}_{i,1}})$.
   - Compute $\mathsf{ek}_{i,1} = \mathsf{S.OcRE.Setup}(1^\lambda, 1^{\mathsf{len}_{i,1}}, \mathsf{crs}_{i,1})$.
   - Compute $\mathsf{msg}_{i,1} = \mathsf{S.OcRE.Encode}(\mathsf{NextMsg}_1, ((\mathsf{x}_i, \mathrm{rand}_i), \perp), 2^\lambda, \mathsf{ek}_{i,1})$.
   - Output $(\mathsf{msg}_{i,1}, r_{i,1}, \mathsf{len}_{i,1})$.
2. **Round 2 ... t:**
   For each subsequent round k, each party P_i does the following:
   - Let τ_{k-2} denote the transcript of the underlying protocol π^{plain} after round $(k-2)$. $\tau_0 = \perp$.
   - Set $\tau_{k-1} = \tau_{k-2}$.
   - For each party P_j, $(j \neq i)$ do the following:
     - Parse its previous round message as $(\mathsf{msg}_{j,k-1}, r_{j,k-1}, \mathsf{len}_{j,k-1})$.
     - Compute $\mathsf{crs}_{j,k-1} = \mathsf{RO}(r_{j,k-1}, 1^{\mathsf{len}_{j,k-1}})$.
     - Compute $\mathsf{msg}^{\mathsf{plain}}_{j,k-1} = \mathsf{S.OcRE.Decode}(\mathsf{msg}_{j,k-1}, \tau_{k-2}, \mathsf{crs}_{j,k-1})$.
     - Append $\mathsf{msg}^{\mathsf{plain}}_{j,k-1}$ to τ_{k-1}.
   - Pick a random string $r_{i,k} \in \{0,1\}^\lambda$. Let $\mathsf{len}_{i,k} = |\mathsf{NextMsg}_k(\mathsf{x}_i; \mathrm{rand}_i, \tau_{k-1})|$.
   - Compute $\mathsf{crs}_{i,k} = \mathsf{RO}(r_{i,k}, 1^{\mathsf{len}_{i,k}})$.
   - Compute $\mathsf{ek}_{i,k} = \mathsf{S.OcRE.Setup}(1^\lambda, 1^{\mathsf{len}_{i,k}}, \mathsf{crs}_{i,k})$.
   - Compute $\mathsf{msg}_{i,k} = \mathsf{S.OcRE.Encode}(\mathsf{NextMsg}_k, ((\mathsf{x}_i, \mathrm{rand}_i), \tau_{k-1}), 2^\lambda, \mathsf{ek}_{i,k})$.
   - Output[13] $(\mathsf{msg}_{i,k}, r_{i,k}, \mathsf{len}_{i,k})$.
3. **Output Computation:**
   Each party P_i does the following:
   - Let τ_{t-1} denote the transcript of the underlying protocol π^{plain} after round $(t-1)$.
   - Set $\tau_t = \tau_{t-1}$.
   - For each party P_j, $(j \neq i)$ do the following:
     - Parse its previous round message as $(\mathsf{msg}_{j,t}, r_{j,t}, \mathsf{len}_{j,t})$.
     - Compute $\mathsf{crs}_{j,t} = \mathsf{RO}(r_{j,t}, 1^{\mathsf{len}_{j,t}})$.
     - Compute $\mathsf{msg}^{\mathsf{plain}}_{j,t} = \mathsf{S.OcRE.Decode}(\mathsf{msg}_{j,t}, \tau_{t-1}, \mathsf{crs}_{j,t})$.
     - Append $\mathsf{msg}^{\mathsf{plain}}_{j,t}$ to τ_t.
   - Output $\mathsf{Out}(x_i, \mathrm{rand}_i, \tau_t)$.

Efficiency of the Protocol:
The size of the messages sent in round k by each party P_i is $3 \cdot \max\{|(\mathsf{msg}_{i,k}, r_{i,k}, \mathsf{len}_{i,k})|\}_{i,k}$. By the definition of strong output-compressing randomized encodings, $|\mathsf{msg}_{i,k}| \leq \mathsf{p}_2(|\mathsf{NextMsg}_k|, |(x_i, \mathrm{rand}_i)|, \log T, \lambda)$ where p_2 is a polynomial. $|\mathrm{rand}_i| = \lambda$, $|\mathsf{NextMsg}_k| = \mathsf{p}_3(|M|)$ where M is the original functionality and p_3 is a polynomial. Also, we know T is at most 2^λ. So $|\mathsf{msg}_{i,k}| \leq \mathsf{p}_3(|M|, |x_i|, \lambda)$ for some polynomial p_3. We know that $|r_{i,k}| = \lambda$ and

[13] Note that to send $\mathsf{len}_{i,k}$, the length of the message is $\log \mathsf{len}_{i,k}$ and so at most λ.

$|\mathsf{len}_{i,k}| \leq \lambda$. Therefore, the size of the messages sent in round k by each party P_i is at most $\mathsf{p}_3(|M|, |x_i|, \lambda)$.

Since π^{plain} is a constant-round protocol, the total communication complexity of our protocol π is at most $\mathsf{p}(n, |M|, |x_1|, \ldots, |x_n|, \lambda)$ for a fixed polynomial p.

6.2 Security Proof

In this section, we formally prove Theorem 6.
Consider an adversary $\mathcal{A}$ who corrupts t parties where $t < n$. Let's say the simulator $\mathsf{Sim}^{\mathsf{plain}}$ for protocol π^{plain} consists of 4 algorithms $(\mathsf{Sim}_1^{\mathsf{plain}}, \mathsf{Sim}_2^{\mathsf{plain}}, \mathsf{Sim}_3^{\mathsf{plain}}, \mathsf{Sim}_{\mathsf{Out}}^{\mathsf{plain}})$ where: $\mathsf{Sim}_1^{\mathsf{plain}}(j, \cdot)$ outputs the adversary's view for the j^{th} of the first t_1 rounds, $\mathsf{Sim}_2^{\mathsf{plain}}$ queries the ideal functionality to receive the output, $\mathsf{Sim}_3^{\mathsf{plain}}(j, \cdot)$ outputs the adversary's view for the j^{th} round of the last $(t - t_1)$ rounds and $\mathsf{Sim}_{\mathsf{Out}}^{\mathsf{plain}}(i, \cdot)$ computes the output of honest party P_i.[14] Also, let's denote the size of $\mathsf{Sim}^{\mathsf{plain}}(\cdot)$ by $s(\lambda)$.

Description of Simulator. The strategy of the simulator Sim for our protocol π against a malicious adversary $\mathcal{A}$ is described below.

1. **Round 1 ... t_1:**
   For each round k and each honest party P_i, Sim does the following:
   - Let τ_{k-2} denote the transcript of the underlying protocol π^{plain} after round $(k-2)$. $\tau_0 = \perp$.
   - Set $\tau_{k-1} = \tau_{k-2}$.
   - For each party P_j, $(j \neq i)$, if $k > 1$, do the following:
     - Parse its previous round message as $(\mathsf{msg}_{j,k-1}, r_{j,k-1}, \mathsf{len}_{j,k-1})$.
     - Compute $\mathsf{crs}_{j,k-1} = \mathsf{RO}(r_{j,k-1}, 1^{\mathsf{len}_{j,k-1}})$.
     - Compute $\mathsf{msg}_{j,k-1}^{\mathsf{plain}} = \mathsf{S.OcRE.Decode}(\mathsf{msg}_{j,k-1}, \tau_{k-2}, \mathsf{crs}_{j,k-1})$.
     - Append $\mathsf{msg}_{j,k-1}^{\mathsf{plain}}$ to τ_{k-1}.
   - Compute $\mathsf{msg}_{i,k}^{\mathsf{plain}} = \mathsf{Sim}_1^{\mathsf{plain}}(k, \tau_{k-1}, \mathsf{st})$ where st denotes the state of $\mathsf{Sim}^{\mathsf{plain}}$.
   - Pick a random string $r_{i,k} \in \{0,1\}^\lambda$.
   - Compute $(\mathsf{msg}_{i,k}, \mathsf{crs}_{i,k}) \leftarrow \mathsf{S.OcRE.Sim}(1^{|s(\lambda|}, 1^{(2\cdot\lambda + |\tau_{k-1}|)}, 1^\lambda, \mathsf{msg}_{i,k}^{\mathsf{plain}}, 1^\lambda)$.
   - Set $\mathsf{RO}(r_{i,k}, 1^{|\mathsf{crs}_{i,k}|}) = \mathsf{crs}_{i,k}$.
   - Output[15] $(\mathsf{msg}_{i,k}, r_{i,k}, |\mathsf{crs}_{i,k}|)$.
2. **Query to Ideal Functionality:**
   Sim queries $\mathsf{Sim}_2^{\mathsf{plain}}(\tau_{k_1}, \mathsf{st})$ and receives an output y in return.
3. **Round $(t_1 + 1)$ ... t:**
   For each round k and each honest party P_i, Sim does the following:
   - Let τ_{k-2} denote the transcript of the underlying protocol π^{plain} after round $(k-2)$. $\tau_0 = \perp$.

[14] $\mathsf{Sim}_1^{\mathsf{plain}}$ also outputs some state that is fed as input to the subsequent algorithms and similarly for $\mathsf{Sim}_2^{\mathsf{plain}}, \mathsf{Sim}_3^{\mathsf{plain}}$.

[15] As before, note that to send the message $|\mathsf{crs}_{i,k}|$, the length of the string is $\log |\mathsf{crs}_{i,k}|$.

- Set $\tau_{k-1} = \tau_{k-2}$.
- For each party P_j, $(j \neq i)$, if $k > 1$, do the following:
  - Parse its previous round message as $(\mathsf{msg}_{j,k-1}, r_{j,k-1}, \mathsf{len}_{j,k-1})$.
  - Compute $\mathsf{crs}_{j,k-1} = \mathsf{RO}(r_{j,k-1}, 1^{\mathsf{len}_{j,k-1}})$.
  - Compute $\mathsf{msg}^{\mathsf{plain}}_{j,k-1} = \mathsf{S.OcRE.Decode}(\mathsf{msg}_{j,k-1}, \tau_{k-2}, \mathsf{crs}_{j,k-1})$.
  - Append $\mathsf{msg}^{\mathsf{plain}}_{j,k-1}$ to τ_{k-1}.
- Compute $\mathsf{msg}^{\mathsf{plain}}_{i,k} = \mathsf{Sim}^{\mathsf{plain}}_3(k, y, \tau_{k-1}, \mathsf{st})$ where st denotes the state of $\mathsf{Sim}^{\mathsf{plain}}$.
- Pick a random string $r_{i,k} \in \{0,1\}^\lambda$.
- Compute $(\mathsf{msg}_{i,k}, \mathsf{crs}_{i,k}) \leftarrow \mathsf{S.OcRE.Sim}(1^{|s(\lambda|}, 1^{(2\cdot\lambda+|\tau_{k-1}|)}, 1^\lambda, \mathsf{msg}^{\mathsf{plain}}_{i,k}, 1^\lambda)$.
- Set $\mathsf{RO}(r_{i,k}, 1^{|\mathsf{crs}_{i,k}|}) = \mathsf{crs}_{i,k}$.
- Output $(\mathsf{msg}_{i,k}, r_{i,k}, |\mathsf{crs}_{i,k}|)$.

4. **Output Computation:**
   Sim does the following:
   - For each honest party P_i, do:
     - Let τ_{t-1} denote the transcript of the underlying protocol π^{plain} after round $(t-1)$.
     - Set $\tau_t = \tau_{t-1}$.
     - For each party P_j, $(j \neq i)$ do the following:
       * Parse its previous round message as $(\mathsf{msg}_{j,k-1}, r_{j,k-1}, \mathsf{len}_{j,k-1})$.
       * Compute $\mathsf{crs}_{j,k-1} = \mathsf{RO}(r_{j,k-1}, 1^{\mathsf{len}_{j,k-1}})$.
       * Compute $\mathsf{msg}^{\mathsf{plain}}_{j,t} = \mathsf{S.OcRE.Decode}(\mathsf{msg}_{j,t}, \tau_{t-1}, \mathsf{crs}_{j,t})$.
       * Append $\mathsf{msg}^{\mathsf{plain}}_{j,t}$ to τ_t.
     - If $\mathsf{Sim}^{\mathsf{plain}}_{\mathsf{Out}}(i, y, \tau_t, \mathsf{st}) = \perp$, send $\perp$ to the ideal functionality and stop.
   - Instruct the ideal functionality to deliver output to the honest parties.

Remarks: Note that if $\mathsf{Sim}^{\mathsf{plain}}$ is a rewinding simulator, our simulator Sim will also be a rewinding simulator.

We include the full proof of indistinguishability in the full version of the paper.

Acknowledgements. The first, second and fourth author's research is supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, and NSF grant 1619348, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C- 0205. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government. The first author's research is also supported in part by the IBM PhD Fellowship. The third author's research is supported by the Israel Science Foundation (Grant No. 468/14), Binational Science Foundation (Grants No. 2016726, 2014276), and by the European Union Horizon 2020 Research and Innovation Program via ERC Project REACT (Grant No. 756482) and via Project PROMETHEUS (Grant 780701). The fifth author's research is supported by NSF CNS-1908611, CNS-1414082 and Packard Foundation Fellowship.

References

[AGVW13] Agrawal, S., Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption: new perspectives and lower bounds. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 500–518. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_28

[AIKW13] Applebaum, B., Ishai, Y., Kushilevitz, E., Waters, B.: Encoding functions with constant online rate or how to compress garbled circuits keys. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 166–184. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_10

[AJ15] Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 308–326. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_15

[AJS17] Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation for turing machines: constant overhead and amortization. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 252–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_9

[AL18] Ananth, P., Lombardi, A.: Succinct garbling schemes from functional encryption through a local simulation paradigm. Cryptology ePrint Archive, Report 2018/759 (2018). https://eprint.iacr.org/2018/759

[AM18] Agrawal, S., Maitra, M.: Functional encryption and indistinguishability obfuscation for turing machines from minimal assumptions. In: TCC (2018)

[BGI+17] Badrinarayanan, S., Garg, S., Ishai, Y., Sahai, A., Wadia, A.: Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10626, pp. 275–303. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_10

[BGJ+17] Badrinarayanan, S., Goyal, V., Jain, A., Khurana, D., Sahai, A.: Round optimal concurrent MPC via strong simulation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 743–775. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_25

[BGJ+18] Badrinarayanan, S., Goyal, V., Jain, A., Kalai, Y.T., Khurana, D., Sahai, A.: Promise zero knowledge and its applications to round optimal MPC. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 459–487. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_16

[BGL+15] Bitansky, N., Garg, S., Lin, H., Pass, R., Telang, S.: Succinct randomized encodings and their applications. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, 14–17 June, pp. 439–448 (2015)

[BL18] Benhamouda, F., Lin, H.: k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 500–532. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_17

[BR93] Bellare, M., Rogaway, P.: Random Oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, CCS 1993, Fairfax, Virginia, USA, 3–5 November, pp. 62–73 (1993)

[BSW11] Boneh, D., Sahai, A., Waters, B.: Functional encryption: definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_16

[CDG+18] Camenisch, J., Drijvers, M., Gagliardoni, T., Lehmann, A., Neven, G.: The wonderful world of global random Oracles. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 280–312. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_11

[CHJV15] Canetti, R., Holmgren, J., Jain, A., Vaikuntanathan, V.: Succinct garbling and indistinguishability obfuscation for RAM programs. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, 14–17 June, pp. 429–437 (2015)

[CIJ+13] De Caro, A., Iovino, V., Jain, A., O'Neill, A., Paneth, O., Persiano, G.: On the achievability of simulation-based security for functional encryption. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 519–535. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_29

[CJS14] Canetti, R., Jain, A., Scafuro, A.: Practical UC security with a global random Oracle. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November, pp. 597–608 (2014)

[DSW08] Dodis, Y., Shoup, V., Walfish, S.: Efficient constructions of composable commitments and zero-knowledge proofs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 515–535. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_29

[GGJS12] Garg, S., Goyal, V., Jain, A., Sahai, A.: Concurrently secure computation in constant rounds. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 99–116. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_8

[GGM86] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986)

[GGMP16] Garg, S., Gupta, D., Miao, P., Pandey, O.: Secure multiparty RAM computation in constant rounds. In: Hirt, M., Smith, A. (eds.) TCC 2016, Part I. LNCS, vol. 9985, pp. 491–520. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_19

[GHL+14] Gentry, C., Halevi, S., Lu, S., Ostrovsky, R., Raykova, M., Wichs, D.: Garbled RAM revisited. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 405–422. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_23

[GOS06] Groth, J., Ostrovsky, R., Sahai, A.: Non-interactive zaps and new techniques for NIZK. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 97–111. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_6

[GS18] Garg, S., Srinivasan, A.: A simple construction of iO for turing machines. Cryptology ePrint Archive, Report 2018/771 (2018). https://eprint.iacr.org/2018/771

[HJK+16] Hofheinz, D., Jager, T., Khurana, D., Sahai, A., Waters, B., Zhandry, M.: How to generate and use universal samplers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 715–744. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_24

[HW15] Hubácek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, 11–13 January, pp. 163–172 (2015)

[HY16] Hazay, C., Yanai, A.: Constant-round maliciously secure two-party computation in the RAM Model. In: Hirt, M., Smith, A. (eds.) TCC 2016, Part I. LNCS, vol. 9985, pp. 521–553. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_20

[IK00] Ishai, Y., Kushilevitz, E.: Randomizing polynomials: a new representation with applications to round-efficient secure computation. In: 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, Redondo Beach, California, USA, 12–14 November, pp. 294–304 (2000)

[IMS12] Ishai, Y., Mahmoody, M., Sahai, A.: On efficient zero-knowledge PCPs. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 151–168. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_9

[IW14] Ishai, Y., Wee, H.: Partial garbling schemes and their applications. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part I. LNCS, vol. 8572, pp. 650–662. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_54

[KLW15] Koppula, V., Lewko, A.B., Waters, B.: Indistinguishability obfuscation for turing machines with unbounded memory. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, 14–17 June, pp. 419–428 (2015)

[LO17] Lu, S., Ostrovsky, R.: Black-box parallel garbled RAM. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 66–92. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_3

[LPST16] Lin, H., Pass, R., Seth, K., Telang, S.: Output-compressing randomized encodings and applications. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016, Part I. LNCS, vol. 9562, pp. 96–124. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_5

[Mia16] Miao, P.: Cut-and-choose for garbled RAM. IACR Cryptology ePrint Archive 2016:907 (2016)

[MW16] Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_26

[Nie02] Nielsen, J.B.: Separating random Oracle proofs from complexity theoretic proofs: the non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_8

[OS97] Ostrovsky, R., Shoup, V.: Private information storage (extended abstract). In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, 4–6 May, pp. 294–303 (1997)

[Pas03] Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 160–176. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_10

[Wee09] Wee, H.: Zero knowledge in the random Oracle model, revisited. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 417–434. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_25