R3PO: Reach-Restricted Reactive Program Obfuscation and its Applications

Kaartik Bhushan¹, Sai Lakshmi Bhavana², Manoj Prabhakaran¹, and Rajeev Raghunath¹

¹ IIT Bombay, Mumbai, India
{kbhushan,mp,mrrajeev}@cse.iitb.ac.in
² IIT (BHU) Varanasi, India
oslbhavana@gmail.com

Abstract. In recent breakthrough results, novel use of garbled circuits yielded constructions for several primitives like Identity-Based Encryption (IBE) and 2-round secure multi-party computation, based on standard assumptions in public-key cryptography. While the techniques in these different results have many common elements, these works did not offer a modular abstraction that could be used across them.

Our main contribution is to introduce a novel notion of obfuscation, called Reach-Restricted Reactive-Program Obfuscation (R3PO) that captures the essence of these constructions, and exposes additional capabilities. We provide a powerful composition theorem whose proof fully encapsulates the use of garbled circuits in these works.

As an illustration of the potential of R3PO, and as an important contribution of independent interest, we present a variant of Multi-Authority Attribute-Based Encryption (MA-ABE) that can be based on (single-authority) CP-ABE in a blackbox manner, using only standard cryptographic assumptions (e.g., DDH) in addition. This is in stark contrast to the existing constructions for MA-ABE, which rely on the random oracle model and supports only limited policy classes.

1 Introduction

Consider the following approach to Identity-Based Encryption (IBE):

– The master key pair is a verification/signing key pair for a signature scheme.
– The decryption key for an identity is simply a signature on the identity.
– The ciphertext is an obfuscation of the following program: it checks if its input is a valid signature on a target identity, and if so, it outputs the message.

With the right notion of obfuscation, as we shall see, this construction indeed translates to a secure IBE scheme! Further, such an obfuscation can be instantiated using standard cryptographic assumptions like DDH, based on the tools in [21, 22].

The motivation of this work comes from the breakthrough results of [7, 18, 22, 27]. These results were surprising not only because of the end results, but also because the central tools involved – garbled circuits, oblivious transfer, smooth projective hash functions, etc. – were all well known for a long time. The power behind these results lay in a machinery that carefully meshed these tools together.

However, this line of works has lacked reusable high-level abstractions, even as the low-level techniques were clearly similar across multiple works. Even the few abstractions of this machinery that appeared subsequently, e.g., in the form of hash-garbling [29], were not comprehensive enough to capture the multifarious applications of the machinery itself.

The main contribution of this work is to develop a versatile abstraction of the common machinery underlying the above works, and take it beyond the current set of applications. Our abstraction involves a strong form of obfuscation, which can be realized for programs that are appropriately sampled. The obfuscation formulation gives an intuitive description of potential solutions, and facilitates realizing it via a novel composition theorem. This not only aids in understanding the current constructions better, but also shows the way to new applications. As an illustration, and as an important contribution of independent interest, we present...
a variant of Multi-Authority Attribute-Based Encryption (MA-ABE) that can be based on (single-authority) ABE in a blackbox manner, using only standard cryptographic assumptions (e.g., DDH) in addition. This is in stark contrast to the constructions available for the original formulation of MA-ABE, which rely on specific assumptions, are for special restricted policy classes and/or are in the random oracle model [17, 19, 43, 50].

1.1 Our Contributions

Our contributions are in two parts – (1) developing a powerful new framework to capture several important results from the recent literature, and (2) using it to construct a multi-authority version of ABE.

The R3PO framework. Our primary contribution is to develop the notion of Reach-Restricted Reactive Program Obfuscation (R3PO) that modularly encapsulates and extends the powerful techniques behind the surprising results of [7, 18, 22, 27, 30]. Our definition of R3PO allows an intuitive description of prior constructions like IBE [22] (with an easy extension to Identity-Based Functional Encryption [59]), 2-Round MPC [7, 27], and RBE [30], all of them using obfuscation of natural reactive programs.

Our applications are instantiated as follows.

• We present a library of useful R3PO schemes. The library includes obfuscations for non-reactive programs that check a commitment opening, verify a signature, and verify the (partial) opening of a hashed value.

• We present a composition theorem that can be used to obtain an R3PO scheme for a reactive program from R3PO schemes for smaller (non-reactive) programs (like those in the library above) into which it decomposes.

• For each of the applications we consider (as well as some of the programs in the library), we define an appropriate reactive program, construct an R3PO for it and use it to complete our construction. The requisite R3PO maybe directly available from the library, or is constructed using the composition theorem.

The garbled circuit technique is entirely encapsulated within the proof of the composition theorem above. This is in contrast with prior work that used these techniques, where the proof would use a sequence of indistinguishability arguments interleaving garbled circuit simulation with other arguments specific to the construction. Indeed, one of the main technical challenges we overcome is to allow disentangling the garbled circuits from the other cryptographic elements, in these security proofs, using a strong simulation based definition of R3PO and our novel notion of decomposition.

Private Multi-Authority ABE Another important contribution of this work is a new version of Multi-Authority Attribute-Based Encryption (called Private Multi-Authority ABE or p-MA-ABE), and a construction for it, conceived in terms of an R3PO.

The motivation for p-MA-ABE stems from the natural use-case for MA-ABE (or even ABE) where a user has privacy requirements against an attribute authority (e.g., they may want to obtain attributes corresponding to a city and a state that they consider their primary home, but without revealing the name of those locations to the authority). Correspondingly, the authority would be willing to issue attributes that satisfy a (possibly private) attribute-granting policy (e.g., issue the attributes for any one state and any one city within that state). The privacy requirement is that the authority (or authorities) shall not learn anything about the attributes of a user, and the user shall not learn anything about the attribute-granting policy, beyond whether the policy is met by the attribute set.

Now, a non-private MA-ABE (or ABE) scheme can be easily converted into a private version, via secure 2-party computation of a function to which the user’s input is their attribute request, and the authority’s input is its master secret key and its attribute-granting policy. Since such a 2PC protocol can be implemented in
two rounds (e.g., a simple protocol based on Yao’s Garbled Circuit works, as we consider the authorities to be honest-but-curious), this only requires the user to send a single message to the server – which we call an attribute request – before the server responds.

p-MA-ABE captures this trade-off: allow the user to initiate the contact with the authority, and in return obtain a strong privacy guarantee. Though the above transformation shows that standard MA-ABE can be easily turned into p-MA-ABE, the former is known to be realizable only for very limited functions and in the random oracle model. In contrast, our results show that p-MA-ABE is as widely realizable as ABE itself!

We give a construction for p-MA-ABE from any (single-authority, ciphertext-policy) ABE scheme in a blackbox manner, using R3PO for (non-reactive) programs for signature checking and commitment opening, that is provably secure in the standard model. The scheme supports general access policies as supported by the underlying ABE scheme, and is policy-hiding if the ABE is policy-hiding.

1.2 Related Work

We mention a few related works below, and discuss how R3PO relate to other notions in Section 2.7.

**Obfuscation.** A large variety of notions of obfuscation have been studied in the literature leading to several important breakthroughs along the way (e.g., [3, 5, 6, 8, 24, 34, 38, 39, 40, 45, 55, 56]). Like R3PO, many of these require security only when the program being obfuscated is generated appropriately [5, 8, 37, 56].

**Composition.** Composition has been considered in the context of cryptographic protocols, leading up to UC security and its variants [4, 13, 14, 15, 20, 23, 35, 49, 57], as well as alternate approaches like Constructive Cryptography [48]. Composition for obfuscation has received far less attention, although it was explicitly considered in an early work [47].

**Garbled Circuits.** Garbled circuits were conceived by Yao [58]. The techniques of chaining multiple garbled circuits appeared in garbled RAM schemes [25, 28, 31, 46], and later several results like Laconic OT [18], IBE from DDH [21, 22], 2-round MPC [7, 27], and several extensions of these works have all relied on these techniques.

**Multi-Authority Attribute Based Encryption (MA-ABE)** The notion of Ciphertext Policy-Attribute Based Encryption (CP-ABE) was introduced in [54] and formally defined in [33]. There is a rich sequence of works realizing ABE, based on lattice based (LWE) [10, 32] and pairing based assumptions [33, 36, 44]. But for MA-ABE, first proposed in [17], realizations so far have been limited. In the standard GID model, [43] formalized the notion of decentralized MA-ABE (where in, no trusted setup algorithm other than a common reference string is allowed), and gave a scheme for it under appropriate bilinear maps assumptions in the random oracle model (supporting general policy structures). A sequence of works culminated in [19], where they gave a (current state-of-the-art) scheme under the Learning With Errors (LWE) assumption in the random oracle model for policies corresponding to DNF formulae. Concurrently, [50] modified the definition to consider sender security (policy hiding) as well as receiver security (attribute hiding), and gave a construction for it under the k-linear assumption in the random oracle model for a special subset of policies.

[42] proposed a variant of MA-ABE called the OT model. It is a relaxed model where there is no global identity fixed for the users. However, as pointed out in [19], this allows multiple users to pool their attributes, defeating one of the main goals of ABE. Our model has a global identity that an authority would incorporate into the key issued for a party, and as captured in the security definition, the user can combine only attributes that are issued for the same global id. Another drawback of [42] was that it used a global setup; we do not. Our setup is local to each authority (as in the global id model). Our model much more closely resembles the standard global id model, but with an additional key request step in the syntax. On the other hand, our results

\[3\] We remark that in a practical situation, this extra round comes at virtually no cost, since anyway a user would first need to establish a secure channel and authenticate itself with the authority before receiving its credentials.
are much stronger than those available in the standard model (which are in the random oracle model and/or for limited function classes). We also offer further flexibility by not requiring each attribute to be attached to a unique authority.

We also note that MA-ABE can be modeled as an appropriate functionality in the framework of public-key Multi-Party Functional Encryption (MPFE) [1]. Their work gives a construction for public-key MPFE for general functionalities. However, this does not yield the result in our work due to the following limitations. Their construction uses an interactive setup, forcing the authorities to be aware of and interact with each other, while we require the MA-ABE authorities to only use “local” setup. Further, their construction is based on Multi-Input Functional Encryption for general functionalities (which is a strong assumption that implies iO). In contrast, we rely only on ABE and standard assumptions. Indeed, the main motivation behind R3PO and the entire line of work leading to it, is to be able to base various cryptographic schemes on simpler assumptions, and to avoid the need for assumptions like iO.

2 Technical Overview

2.1 Motivating Examples

We start with a few motivating constructions, along the lines of the IBE construction mentioned at the beginning of this paper, which we seek to base on our new notion of obfuscation. In the general case, we would be obfuscating a reactive program (or more specifically, a Moore machine), which at each step, accepts an input, updates its state, and produces an output based on the new state.

Identity-Based Functional Encryption. IBFE is an extension of IBE where each identity \( id \) is associated with a unique function \( f_id \) (not known to the encryptor), so that when an encryption of a message \( m \) addressed to \( id \) is decrypted using the key for \( f_id \), one receives \( f_id(m) \). An IBFE scheme can be obtained by simply modifying the IBE scheme above so that the obfuscated program takes a signature on \((id, f)\) (where \( id \) is already fixed in the program, but \( f \) is not), and transitions to a state encoding \( f \), where it outputs \( f(m) \).

IBFE has been explored in a prior work [59], but their definition is incomparable to our notion above. On the one hand, their definition does not allow the adversary to obtain any function keys – under any IDs – for a function \( f \) such that \( f(m_0) \) and \( f(m_1) \) are not equal; on the other hand, it is not made very clear if the adversary is restricted to obtaining a single function key for the challenge \( id \), as is the case in our definition. Finally, they offer a construction for the primitive only for a very restricted class of functions, while our construction supports general functionalities.

2-Round MPC. Following the constructions in [7, 27], an underlying (multi-round) MPC protocol can be reinterpreted as evaluating a blinded circuit, in which each boolean gate is owned by a party, and the protocol amounts to evaluating the wires of this circuit publicly. The wire values are public, but each gate is private to its owner.

The 2-round MPC constructed from the blinded circuit is as follows. In the first round, each party broadcasts a commitment to the 4 bits (separately) of the truth table of each of its gates. In the second round, each party broadcasts the obfuscation of the following reactive program:

- The program maintains a public state consisting of all the wire values of the circuit, evaluated thus far.
- If the next gate is owned by another party, the program accepts as input the output wire value of the gate, along with an opening of the corresponding commitment in the gate. If the opening verifies, it updates its state to correspond to having evaluated this wire. It produces no output for this transition.
- If the next gate is owned by this party, then it takes no input, transitions to a state that includes the output wire value of this gate, and outputs the opening of the corresponding commitment.
Finally, given these obfuscated reactive programs, the parties evaluate the blinded circuit gate by gate, at each step first running the program from the owner of the gate, and then feeding its inputs to all the other programs.

Laconic OT. This is a version of OT in which the receiver has a vector $D$ of choice-bits, which it commits to by sending a short string $y$ to the sender. Later, on input $(i, x_0, x_1)$, the sender should send a string to the receiver from which the latter should learn only $x_{D_i}$.

We consider the following implementation of Laconic OT: Using a hash that supports “selective opening” of a bit in the hashed string, with a collision resistance guarantee that prevents opening any bit in two different ways, the receiver hashes $D$ to obtain $y$. On input $(i, x_0, x_1)$, the sender obfuscates the following (small) program and sends it over to the receiver: The program accepts as input an opening of $y$ at position $i$ to a bit $b$, and if the opening is valid, then it outputs $x_b$.

Each of the above simplistic constructions relied on an intuitive notion of “obfuscation.” In the sequel, we develop a formal notion of obfuscation which will let us make the above descriptions precise, while retaining their simplicity. Importantly, our new obfuscation notion is indeed realizable in all the above cases, using the same standard cryptographic assumptions as in the prior works which introduced these constructions.

2.2 Defining R3PO

At a high-level, we consider obfuscation of reactive programs. A reactive program (a finite-state machine, or more precisely, a Moore machine) takes inputs over multiple rounds, updating its state and producing an output based on the state at each round. It is specified by a start state, a transition function $\pi$ and a message function $\mu$, so that, on reaching a state $\sigma$, the program outputs $\mu(\sigma)$.

Before discussing the definitions, it will be useful to have a couple of running examples in mind. In these examples, $\mu$ is arbitrary (and secret), and a public $\pi$ is as specified below.

- **Commitment.** $\pi_c$ incorporates a commitment string $c$. On input $d$ at the start state, if $d$ decommits $c$ to $m$, then $\pi_c$ transitions to a state $\sigma_m$ encoding $m$.

- **Signature.** A signature verification key $vk$ is encoded in the start state $\sigma_{vk}$ of $\pi$ (denoted as $\pi[\sigma_{vk}]$), from where, given a valid signature on a message $m$ as input, it transitions to a state $\sigma_m$ encoding $m$.

These are both instances of “one-step programs” which have transitions only out of the start state. (We shall later explain the slightly different choices for how the values $c$ and $vk$ are incorporated into $\pi$ in the two cases.) In these examples, $\pi$ is not hidden, and the goal of obfuscating such a program would be to hide $\mu$. More generally, $\pi$ and $\mu$ can both have secrets in them (when defining reactive programs formally, we will denote them as $\pi^{(\alpha)}$ and $\mu^{(\beta)}$, where $\alpha$ and $\beta$ are the secrets).

Reach Extraction and Simulation. Our simulation-based notion of obfuscation requires that a “reach-extractor” should exist for the program being obfuscated. A reach extractor would predict all the states of a reactive program that are reachable using inputs that can be efficiently computed by any adversary. Then, the obfuscation of the program should be simulated using only the outputs produced by the program at those states. We elaborate on reach-extraction and the rest of the simulation below.

Reach Extractability. Which states in a program $\pi$ are efficiently reachable is a consequence of the process that generates the program (analogous to how an “evasive program” being evasive is a consequence of sampling it from a distribution). This process involves a generator $G$ and an adversary $Q$. A reach extractor for an adversary $Q$ is a program that passively (possibly in a non-blackbox manner) observes $Q$ as it interacts with $G$, and then predicts (a superset of) the set of states that the adversary will be able to reach in the program output by $G$. This prediction is made explicitly in the form of inputs to a (possibly different) reactive program $\Pi$ that will reach all the states reachable by the adversary, and perhaps more. Here we allow the extractor to specify $\Pi$, which belongs to a transition function family $\mathcal{P}$ that may be different from
the transition program family \( P \) that is obfuscated. We refer to this as the “reach bounding” guarantee of the reach extractor.

We illustrate a reach extractor for the two running examples.

– **Commitment:** \( G \) accepts a commitment string \( c \) from \( Q \), and then outputs \( \pi_c \). A reach extractor can extract a value \( m \) from the commitment, either when \( Q \) is semi-honest, or when a setup is used that the extractor can control. Now, \( m \) is not a decommitment as expected by \( \pi_c \). Instead, we allow the extractor to specify a different program \( \Pi_m \) which accepts \( m \) itself as the input and transitions to \( \sigma_m \). This extractor is reach bounding, because, due to the binding property of the commitment scheme, the only state \( Q \) could reach in \( \pi_c \) is also \( \sigma_m \).

– **Signature:** In this case, \( G \) internally samples a pair \((sk, vk)\) of signing and verification keys. It sends \( vk \) to \( Q \), and further may answer signature requests by \( Q \). An extractor can collect all the signatures \( Q \) receives from \( G \) and output them as a reach-bounding set of inputs for \( \Pi = \pi[\sigma_{vk}] \). Note that here the program family to be used by the extractor \( \mathcal{P} \) is the same as the one being obfuscated \( P \) (and it has only one program in it but with various start states). The reach bounding property follows from unforgeability of the signature scheme.

**Simulation.** An obfuscator for a generator R3PO security definition requires that a 2-stage simulation exists for any adversary \( Q \), as follows:

– **Stage 1:** After \( Q \) finishes interacting with \( G \), a reach extractor observing \( Q \) specifies a set of reachable states (in the form of a program \( \Pi \) and inputs to it).

– **Stage 2:** Given the output of the original message function \( \mu \) on those states, a simulated obfuscation is produced. This should be indistinguishable from the obfuscation of the reactive program produced by \( G \), even given auxiliary information output by both \( G \) and \( Q \).

Note that this is a stronger notion of simulation than even VBB obfuscation, which only requires the simulation of one predicate at a time, rather than a simulation of the entire obfuscated program. Indeed, requiring such a simulator would typically entail that the program is learnable and hence trivial to obfuscate. What keeps our definition from becoming trivial is the fact that the extracted inputs are a function of the program generation process, and are not available to the obfuscator.

**Reach Restriction.** The final component in our definition of R3PO is in the form of an additional requirement on the reach extractor in Stage 1 above. This requirement stems from the “one-time” nature of Yao’s Garbled Circuits, a key ingredient in the constructions that we wish to capture. Intuitively, these constructions require that an adversary can evaluate any garbled circuit on only one set of inputs. We incorporate a corresponding reach restriction requirement into our definition of reach extractability of a reactive program (Definition 3), which leads to the name reach-restricted reactive programs (R3P). \(^4\)

To define reach restriction, we require the state space of the reactive programs to be a priori partitioned into a polynomial number of parts, \( \Sigma = \Sigma_1 \cup \cdots \cup \Sigma_N \). Then, informally, the reach restriction property of a reactive program is that no efficient adversary would be able to find inputs that take \( \pi \) to two different states that belong to the same part. Formally, the reach restriction property is imposed on the reachable states produced by the reach-extractor.

We return to our running examples.

– **Commitment:** We let \( \Sigma_1 \) consist only of the start state and \( \Sigma_2 \) consist of all states of the form \( \sigma_m \). Since the extractor outputs only one message \( m \), the reach restriction property already holds.

– **Signature:** We let \( \Sigma_1 \) consist of all the potential start states \( \sigma_{vk} \) and \( \Sigma_2 \) consist of all states of the form \( \sigma_m \) (the two kind of states are encoded so that \( \Sigma_1 \cap \Sigma_2 = \emptyset \)). To be reach restricting, we will require that the generator \( G \) gives out at most one signature. Further, we would want to enforce that breaking reach

\(^4\) Formally, we do not define R3P, but only an R3P Generator, as a program generator (Definition 2) that has a reach extractor.
restriction in \( \Pi \) must correspond to forging signatures with respect to the key sampled by \( G \). This is enforced by keeping \( \nu_{vk} \) in the start state of \( \pi[\nu_{vk}] \) (rather than in the transition function itself), which in turn forces \( \Pi \) to use the same start state and hence the same verification key.

### 2.3 R3PO Composition Theorem

As noted earlier, a major motivation of this work is to encapsulate a range of powerful techniques using garbled circuits in a reusable form. This result takes the form of a composition theorem, which allows obfuscating a reactive program via an obfuscation of its various components.

The high-level idea is to view a reactive program \( \pi \in P \), over a state space \( \Sigma = \Sigma_1 \cup \cdots \cup \Sigma_N \) as consisting of separate programs \( \pi_1, \ldots, \pi_N \), such that \( \pi_i \) is identical to \( \pi \) on states \( \sigma \in \Sigma_i \), and in other states it ignores all inputs (i.e., remains at the same state). Let \( P_i \) denote the class of such programs \( \pi_i \). W.l.o.g. (due to reach-restriction), we require \( \pi \) to not have any transitions between states in the same part, and hence each \( \pi_i \) is a “one-step” (or non-reactive) program that halts after its first transition out of the start state. However, attempting to formalize this leads to a couple of conundrums.

**Conundrum 1: Dynamically Determined Programs.** As a naïve starting point, one could try building an obfuscator for \( P \) from obfuscators for \( P_i \). However, this runs into an immediate problem: When executing a program in \( P \), the state reached in \( \Sigma_i \) is dynamically determined by the inputs used, whereas when obfuscating a program in \( P_i \), its start state needs to be fixed. The resolution of this conundrum, which goes back to [18, 21, 22, 25, 28, 31, 46], is to provide a garbled circuit that can dynamically compute the obfuscation of \( \pi_i[\sigma_i] \) with the correct start state \( \sigma_i \); the input to this garbled circuit would be the labels encoding \( \sigma_i \), which in turn would be released by the obfuscation of a previous program \( \pi_j[\sigma_j] \) on an input \( x \) such that \( \pi_j(\sigma_j, x) = \sigma_i \).

However, the price we pay for using garbled circuits is that only one set of labels can be made available to the adversary for each garbled circuit, in turn resulting in the reach-restriction requirement.

**Conundrum 2: Intertwined Generators.** Recall that to formalize reach restriction, our definition needed to take into account the generators. Now, when we try to map the different parts of a single reactive program as being generated by multiple generators, the generators can become deeply intertwined, sharing secret keys and state variables. Further, the program generated by one generator needs to have a start state that is determined by the outputs produced by programs in other parts. So it may not always be possible to view a (reach-restricted) reactive program produced by a generator as the composition of single-step reactive programs produced by separate generators.

The resolution to this conundrum is to require some additional relation between the generator for the reactive program and the generators for the one-step programs. This leads us to the notion of decomposition.

**Decomposition.** Unlike in the case of MPC protocols, wherein the subprotocols are explicitly executed by a composite protocols, a reactive program generator need not have “sub-generators” running within it. Indeed, this presents a challenge to composition that is fundamentally different from composition in MPC.

Our novel solution is to define decomposition in terms of a bisimulation requirement. Roughly, for \( G \) to decompose into a smaller generator \( H \) (and additional computation), we require that \( G \) can be viewed as \( H \) via a simulator, and vice versa. More precisely, we require that there be two simulators \( J \) and \( Z \) such that 

\[
\begin{array}{c}
\begin{array}{c}
\sum \pi \\
\bot \pi \\
J \\
\end{array}
\end{array}
\] (denoting that \( J \) internally runs \( G \) as a black box) and 

\[
\begin{array}{c}
\begin{array}{c}
\sum \pi \\
\bot \pi \\
Z \\
\end{array}
\end{array}
\] are indistinguishable from each other from the point of view of any adversary \( Q \) (or more precisely, for \( \sum \pi \), where the wrapper \( W \) is also part of the simulation). This by itself can be trivially arranged by letting \( J = H \) and \( Z = G \). We need to further capture the requirement that the program \( \pi \) produced by \( H \) corresponds to a single step in the program \( \pi \) produced by \( G \). More precisely, the state space of the generator \( H \) corresponds to a part \( \Sigma_i \) of the state space of \( G \), and we require that the start state of \( \pi \) is the same as the only state in \( \Sigma_i \) that is reachable in \( \pi \).
Now, by requiring this, we require $J$ to know the reachable state in $\pi$ produced by $G$. While this is possible in some cases (e.g., when the reachable state is determined by a signed message sent by $G$), in certain other cases it is not possible (e.g., when it is determined by a message hidden in a commitment). To accommodate these different situations, we allow $J$ to obtain this information from the wrapper $W$, which is in turn allowed to obtain this from a reach-extractor for $G$ (or more precisely, from a “partial” reach extractor which only extracts the reach within $\Sigma_i$).

Finally, for use in our composition theorem, we shall require a uniformly sampled message function to be associated with the reactive program produced by $J$. (While the definition of decomposition allows arbitrary message function class here, the composition theorem is for decomposition that uses a particular message function class.)

We refer the reader to Section 5.1 for a more detailed discussion and a precise definition of decomposition.

Composition. Having defined decomposition, we turn to stating and proving the composition theorem. Informally, it states that if a generator $G$ decomposes into generators $(H_1, \ldots, H_N)$ (for a partition of its state space $(\Sigma_1, \ldots, \Sigma_N)$), and if each $H_i$ has an R3PO scheme $O_i$, then there is one for $G$ as well. The construction uses garbled circuits, following the outline at the beginning of this section. The final obfuscation consists of one garbled circuit $GC_i$ for each part $\Sigma_i$, such that on reaching $\sigma \in \Sigma_i$, an evaluator would have the labels that encode $\sigma$ as input for $GC_i$, and $GC_i$ would then output $\mu(\sigma)$ as well as an obfuscation $O_i(\hat{\pi}_i[\sigma], \hat{\mu}_i)$ (using a hard-coded random tape). Feeding an input $x$ to this obfuscated program will release the labels for the state $\hat{\pi}_i(\sigma, x) = \pi(\sigma, x)$.

To prove that this construction yields an R3PO for $G$, we use a sequence of hybrids that would replace one garbled circuit at a time with a simulated one, which in turn outputs not the actual obfuscation $O_i(\hat{\pi}_i[\sigma], \hat{\mu}_i)$, but a simulated one. At each step, we will be able to apply the decomposition guarantee (using an inductively maintained partial reach extractor) to go from $G$ to $\frac{\Sigma_i}{J_i}$, wherein we use the R3PO guarantee to replace the actual obfuscation used to simulate $GC_i$ with a simulated one (while also extending the partial extractor); then we move back from $\frac{\Sigma_i}{J_i}$ to $\frac{\Sigma_i}{J_i}$ and then $G$.

2.4 R3PO Library

We present R3PO schemes for a few basic program classes which can be combined together in a variety of constructions.

– Commitment-Opening. This is similar to the running example presented above. In Section 4.1, we realize the R3PO for a couple of flavors of this (UC secure commitment, and “weakly secure” commitment that is suitable for semi-honest committers), based on the standard assumption of 2-round OT.

– Signature-Checking. We provide an R3PO for signature-checking programs as in the running example. To facilitate full security in applications like IBE and IBFE, we support puncturable signature schemes. We instantiate a puncturable signature scheme and give an R3PO scheme for this program family assuming an OTSE scheme in Section 4.2.

– Hash-Opening. This is similar to the commitment opening reactive program, but with a compressing hash instead of a binding commitment. The R3PO for this program class can be constructed from Laconic OT [18]. Alternately, we can use our composition theorem to bootstrap from an R3PO for the same class instantiated with a factor-2 compressing laconic OT (see Section 2.5 below).

5 In our constructions of IBE and IBFE, the ciphertext corresponds to an obfuscated program. For full security, the adversary must be allowed to make key queries even after receiving the ciphertext. But, no interaction is allowed between the program generator and the adversary after the program has been generated. Hence, we consider a generator which gives out an appropriately punctured signing key before the interaction finishes.
\(\epsilon\)-Transition While specifying reactive programs using the above building blocks, often it is useful to transition from state reached via one building block to a state that is suitable as the start state of another building block. \(\epsilon\)-transitions provide the essential syntactic sugar to enable this. R3PO for an \(\epsilon\)-transition is implemented using a garbled circuit.

2.5 Applications: The Different Ways of Using R3PO

Our R3PO library and our composition theorem form a versatile toolkit for instantiating new and old constructions. There are a few different ways in which they can be put to use.

**Off-the-Shelf Without Composition.** In certain cases, the components in our library are already powerful enough off-the-shelf to yield a construction for a desired application. An illustrative example is that an R3PO for (puncturable) signature-checking can be used to construct an IBFE scheme (and, as a special case, IBE), as sketched in Section 2.1. The security proof is fairly direct, by using a generator for the R3PO that models the security experiment of IBFE.

**Using Composition.** We illustrate a typical “workflow” for using the R3PO composition theorem in a higher-level application. We use the example of Laconic OT [18], which is one of the early constructions that form the inspiration for this work. For the sake of readability we use slightly imprecise terminology.

\[\text{– We start by identifying a reactive program family, such that an R3PO for it directly yields our application.}^{6}\]

In the case of laconic OT, this reactive program traverses a pre-determined path along a Merkle tree, with states holding the hash value at each node, and making a transition if the input “explains” the hash at that node. The Merkle tree uses an underlying hash scheme which compresses by a factor of 2.

\[\text{– We consider the one-step restrictions of this reactive program as another reactive program family, and carry out the following two steps:}\]

\[\bullet \text{ We show that the original reactive programs can be decomposed into its one-step restrictions. This involves matching the definition of decomposition with straightforward constructions.}\]

\[\bullet \text{ We give an R3PO scheme for these one-step restrictions. This can be directly based on the construction in [18] for factor-2-compression laconic OT (not involving garbled circuit chaining).}\]

\[\text{– Then we simply invoke our composition theorem to obtain an R3PO for the original program family.}\]

\[\text{– We package the original reactive program as another one-step program, so that it can be included in our R3PO library for various applications (see Appendix C.3). Laconic OT is a direct consequence of an R3PO for this one-step program.}\]

Another example of this workflow is in the construction of the R3PO for signature-checking that was mentioned above as part of our library (where the smaller non-reactive programs used correspond to one-time signature checking).

**R3PO as a Component.** In the above examples, once an R3PO is constructed, the final application is fairly immediately realized. However, it is also possible to use R3PO as a component in a larger construction. One such example is the 2-round MPC protocol of [7, 27], which we rederive in Appendix F using R3PO for commitment opening. In this construction, each party publicly makes several commitments in the first round, and then each of them publicly posts an R3PO for a program (which in turn decomposes into commitment opening programs); the final computation involves running all these obfuscated programs mimicking an underlying MPC protocol. Here, the security of R3PO can be used to move to an “ideal” execution of the

---

6 For complex applications, the step from R3PO to the final security proof may be non-trivial. The proof may involve multiple hybrids, with R3PO security used to replace a real obfuscation in one hybrid with the simulated obfuscation in the next. This is the case for applications like 2-round MPC and p-MA-ABE below.
MPC protocol (or more precisely, build a simulator for the 2-round MPC using the simulators of R3PO and the simulator for the underlying MPC protocol).

Our main application of p-MA-ABE (discussed below) also falls into this category, where the security of the final construction depends on several components, one of which is an R3PO scheme. This construction also illustrates the possibility of combining multiple library components (commitment-opening and signature-checking) in the same reactive program.

2.6 Private Multi-Authority ABE

In this section, we give a brief overview of the new variant of MA-ABE that we introduce, called Private Multi-Authority ABE (p-MA-ABE), and the main ideas behind our construction for it. Our construction is intuitive in terms of an obfuscation of a reactive program, and can indeed be realized using R3PO. The flexibility of the new framework allows a relatively easy construction, using existing ABE schemes, and with a robust security definition. The full description can be found in Section 7.

Defining p-MA-ABE: The setting of p-MA-ABE (as well as MA-ABE) involves a set of mutually distrusting authorities (say $A_1, \ldots, A_N$), a sender and a receiver. The algorithms in a p-MA-ABE scheme are as follows:

- Setup: At the start of the execution, each authority $A_i$ does a local (decentralized) setup to generate its public and secret keys $(mpk_i, msk_i)$ and shares the public key $mpk_i$ with the other users in the system.
- Key-Request: a receiver can construct a set of key-requests $req = (req_1, \ldots, req_N)$ for a global identifier $gid$ and attribute set $\bar{x}$ from the public keys. It can then submit a key-request query (of the form $(gid, req_i)$) to an authority $A_i$ and get back a key-component $sk_{gid, req_i}$ from $A_i$ ($req$ will hide $\bar{x}$).
- Key-Gen: an authority $A_i$ receives as input a key-request $req_i$ for a global identifier $gid$, and outputs a key-component $sk_{gid, req_i}$ that incorporates an attribute-granting policy $\Theta_i^{gid}$ (which, for simplicity, we do not consider a secret).
- Encryption: a sender can encrypt a message $m$ with a ciphertext policy $\phi$, using the public keys of the authorities to produce a ciphertext $ct_{m,\phi}$.
- Decryption: a receiver can decrypt a ciphertext $ct_{m,\phi}$ using key components of the form $(sk_{gid, req_1}, \ldots, sk_{gid, req_N})$ where all the requests $req_i$ were generated using $gid$ and $\bar{x}$ such that $\Theta_i^{gid}(\bar{x}) = 1$ for all $i$, and $\phi(\bar{x}) = 1$.

Compared to the original definition of MA-ABE, there are two main differences in p-MA-ABE: Firstly, since the attributes are to be kept private even from the authorities, there is a key-request step, wherein the user generates the key-request messages to all the authorities based on its desired set of attributes. Secondly, we allow each authority to use an arbitrary attribute-granting policy, which depends on the entire attribute vector.\footnote{In particular, it captures the standard formulation of MA-ABE, where each authority $A_i$ “owns” a set of attributes and its key-issuing decision is based on the values of the attributes it owns.}

We define security w.r.t. a corruption model where the adversary is allowed to maliciously corrupt the receivers and semi-honestly corrupt any subset of authorities. If a receiver is honest, we require that the key-request req reveals nothing about $\bar{x}$ to the authorities, even if all of them collude. When the receiver is corrupt, we guarantee that, for any choice of $(\phi, m_0, m_1)$, the adversary cannot distinguish between the encryptions of $m_0$ and $m_1$ w.r.t. a policy $\phi$, unless for a pair $(gid, \bar{x})$ such that $\Theta_i^{gid}(\bar{x}) = 1$ for all honest authorities $A_i$, $\phi(\bar{x}) = 1$ and the adversary sent a valid key request for $(gid, \bar{x})$ (i.e., a request that can be produced by the Key-Request algorithm on those inputs) to at least one honest authority (and it could have sent it to the others as well).
A p-MA-ABE Scheme: Our scheme is easily described in terms of obfuscating a reactive program. The key-request reqᵢ is a commitment to ¯x (using a common random string in the public-key of Ai). The key issued by each authority is the obfuscation of a reactive program; the reactive programs by the different authorities “talk” to each other and confirm that they all agree on granting the same attribute ¯x to gid, and if so, issue standard CP-ABE keys for ¯x. More precisely, the reactive program (π(α), µ(β)) works as follows (with Ai’s CP-ABE master secret-key constituting the secret β; α can be empty, or alternately, can be used to store Θ gid_i privately):

– at the start state accepts a decommitment for reqᵢ and transitions to a state with ¯x. There, if Θ gid_i(¯x) = 1, then it outputs a signature on (gid, ¯x), using Ai’s signature key.
– Then, it moves through N – 1 states accepting signatures on (gid, ¯x) from all the other servers.
– On reaching the last of these states, it outputs a CP-ABE key for the attribute ¯x, under a (standard) CP-ABE scheme for which Ai is the authority.

If Θ gid_i(¯x) = 1 for all i, then the receiver can obtain the CP-ABE keys for ¯x under all the authorities. Now, to encrypt a message under a policy φ, one simply secret-shares m into N shares, and encrypts each share under the CP-ABE public key of the corresponding authority.

Using the composition theorem, we show that there is an R3PO for a suitably defined generator that models the p-MA-ABE security game. (As it turns out, we need to do this for two different generators to handle two different hybrids; the first hybrid does not rely on the unforgeability of the signatures, and lets the adversary specify all the signing keys. We show that the same obfuscator O is an R3PO scheme for both the generators.)

Finally, we mention that given the above construction, in hindsight, it is possible to modify it into one that does not rely on R3POs, and directly uses garbled circuits, OT and IBE (along with CP-ABE). This is not entirely surprising since, when our construction is unrolled, it uses only R3POs for commitment and signature (which are related to OT and IBE, respectively), and our composition theorem (based on garbled circuits). However, the ability to visualize the scheme in terms of obfuscation was instrumental in discovering it.

2.7 Comparison of R3PO with Existing Primitives

It is instructive to compare R3PO with various existing primitives and techniques.

Hash Garbling. This abstraction from [29] gives a similar interface to R3PO for a specific class of program generators, namely, “Hash Opening.” More precisely, hash garbling involves a hash-opening check as well as a circuit evaluation, which corresponds to a reactive program that carries out a hash-opening transition followed by an epsilon transition (that evaluates the circuit).

{Batch, Hash, Chameleon, OneTime Signature}-Encryption. These flavors of encryption that were introduced in prior work [11, 21, 22] correspond to R3PO schemes for one-step programs that are included in our library (Hash Opening and Signature Checking). While these original definitions differ in their details, R3PO provides a simulation-based definition that can be uniformly used in all their applications.

Witness Encryption over Commitments (cWE). cWE, recently defined in [12], is quite similar to an R3PO for “Commitment Opening” followed by an epsilon transition. It was instantiated from Oblivious Transfer (OT) and garbled circuits, just as the R3PO scheme obtained directly from our library and the composition theorem is.
**Garbled Circuit Chaining.** The technique of garbled circuit chaining has appeared in a long line of works [7, 22, 27, 30, 46]. We note that R3PO allows different one-step programs (for example combining commitment and signature in p-MA-ABE), while all prior works used garbled circuit chaining with links that correspond to a single cryptographic element. Also, as already mentioned, the prior works do not separate out the chaining from the cryptographic elements that are chained together.

**Obfuscation notions.** Our notion of R3PO is different in many ways from the other notions of obfuscation. Many notions of obfuscation are either unrealizable in general or inhabit “obfustopia,” requiring a combination of relatively strong assumptions, and are not practical in terms of efficiency [3, 5, 6, 8, 16, 38]. But there are a few exceptions for specialized applications, like obfuscation of reencryption based on bilinear pairings [37] or compute-and-compare obfuscation based on LWE [56]. R3PO could be considered to be in the latter group, but with a much richer class of applications compared to the others.

2.8 Future Directions

As the R3PO framework has several new elements, in this work we have not ventured into many useful extensions, nor attempted to exploit all the available features. Several directions suggest themselves, for future work: 1. Extending the framework, by allowing the exact states during transition to be hidden. 2. Expanding the library, with new primitives, and to capture more constructions, like that of Garbled RAM [46] as instances of R3PO. 3. Capturing additional techniques used in extensions of the IBE or 2-round MPC protocol [9, 26] under the umbrella of R3PO. 4. Constructing more applications by exploiting the full power of the R3PO framework and novel combinations of the library elements.

3 Defining Reach-Restricted Reactive Program Obfuscation

In this section we present the formal definitions that were motivated in Section 2.2.

3.1 Reactive Programs and Generators

Below we define a reactive program as a stateful machine that takes inputs, transitions its state and produces outputs as a function of its state. Formally such a program consists of a deterministic transition function $\pi$ and a deterministic message function $\mu$, both of which can be parameterized by (secret) values $\alpha$, $\beta$ (hardwired into circuits $\pi(\alpha)$, $\mu(\beta)$ respectively).

**Definition 1 (Reactive Program over ($\mathcal{X}, \Sigma, A, B, M$)).** A reactive program $(\pi^{(\alpha)}, \mu^{(\beta)})$, with input alphabet $\mathcal{X}$, a state-space $\Sigma$, a start-state start $\in \Sigma$ and secret spaces $A, B$ is specified by a deterministic transition program $\pi^{(\alpha)} : \Sigma \times \mathcal{X} \to \Sigma$ parameterized by a secret $\alpha \in A$ and a deterministic message function $\mu^{(\beta)} : \Sigma \to M$ parameterized by a secret $\beta \in B$, that on input sequence $(x_1, \ldots, x_\ell) \in \mathcal{X}$, reaches a state $\sigma_\ell$, where $\sigma_i = \pi^{(\alpha)}(\sigma_{i-1}, x_i)$ for $i = 1, \ldots, \ell$ and $\sigma_0 = \text{start}$, and outputs a message $\mu^{(\beta)}(\sigma_\ell)$. We also define $\text{REACH}_{\pi^{(\alpha)}}(x_1, \ldots, x_\ell) = \{\sigma_0, \ldots, \sigma_\ell\}$ and $\pi^{(\alpha)}(x_1, \ldots, x_\ell) = \sigma_\ell$.

Reactive programs have an associated implicit security parameter $\kappa$: specifically, we require that the states in $\Sigma$ and secrets in $A, B$ are represented as binary strings of length polynomial in the security param-

---

8 As described in Appendix B.1, restricting our obfuscations to deterministic message functions is without loss of generality, even if we are interested in randomized message functions.
self-transition \( \kappa \), and the functions \( \pi^{(\alpha)} \) and \( \mu^{(\beta)} \) are polynomial in \( \kappa \). Throughout the rest of the paper, we shall omit \( \kappa \) and implicitly refer to “polynomial in \( \kappa \)” as simply being polynomial.

**Partition Function and Program Class.** A transition function class \( \mathcal{P} \) refers to a set of transition functions along with an associated partition function \( \mathcal{I} \) that maps states to integers, i.e., \( \mathcal{I} : \Sigma \rightarrow \{1, \ldots, N\} \) for some positive integer \( N \). We say that \( \mathcal{I} \) partitions the state space \( \Sigma \) into \( \Sigma_1, \ldots, \Sigma_N \) where,

\[
\Sigma_i = \mathcal{I}^{-1}(i) := \{ \sigma \mid \sigma \in \Sigma, \, \mathcal{I}(\sigma) = i \}
\]

Unless otherwise stated, the start state of a reactive program is assumed to be in \( \Sigma_1 \). We say that a transition function \( \pi^{(\alpha)} \) is **tree-ordered** with respect to \( \mathcal{I} \), if the directed graph over \( \{1, \ldots, N\} \) (each partition as a vertex) with an edge-set

\[
\{(i, j) \mid \exists \text{ distinct } \sigma \in \Sigma_i, \, \sigma' \in \Sigma_j, \, \exists x \in \mathcal{X} \text{ s.t } \pi^{(\alpha)}(\sigma, x) = \sigma' \}
\]

is a tree, and all its edges \((i, j)\) satisfy \( i < j \). That is, for any partition \( j \), there is at most a single partition \( i < j \) from which states in partition \( j \) can be transitioned to. Further, we say that a transition function class \( \mathcal{P} \) is tree-ordered if every \( \pi^{(\alpha)} \in \mathcal{P} \) is tree-ordered w.r.t. the partition associated with \( \mathcal{P} \).

A program class \((\mathcal{P}, \mathcal{M})\) is a set of reactive programs \((\pi^{(\alpha)}, \mu^{(\beta)})\) with \( \pi^{(\alpha)} \in \mathcal{P} \) and \( \mu^{(\beta)} \in \mathcal{M} \).

**Reactive Program Generator.** We now describe the process which generates a reactive program to be obfuscated. A PPT program \( G \) (which we call the generator) interacts with a PPT program \( Q \) (which we call the adversary) over many rounds; at the end \( G \) outputs a reactive program \((\pi^{(\alpha)}, \mu^{(\beta)})\). Both \( G \) and \( Q \) are also allowed to produce auxiliary outputs. The goal of obfuscation will be to hide the secrets \( \alpha, \beta \) from the adversary, except for the outputs it can get from the states that it reaches in the program \( \pi^{(\alpha)} \).

**Definition 2 ((\mathcal{P},\mathcal{M})-Generator \( G \)).** A \((\mathcal{P}, \mathcal{M})\)-**generator** \( G \) for a transition function class \( \mathcal{P} \) and message function class \( \mathcal{M} \) is a PPT interactive program that interacts with an arbitrary PPT program \( Q \). We write

\[
\left( (\pi^{(\alpha)}, \mu^{(\beta)}), a_G; \, a_Q \right) \leftarrow (G : Q)
\]

to indicate that at the end of the interaction, \( G \) outputs \((\pi^{(\alpha)}, \mu^{(\beta)}, a_G)\) and \( Q \) outputs \( a_Q \) (where \( \pi^{(\alpha)} \in \mathcal{P} \), \( \mu^{(\beta)} \in \mathcal{M} \)). A **generator class** is simply a set of generators.

We shall also consider an adversary class \( Q \). Depending on the application, we consider \( Q \) to contain all PPT machines, or only “semi-honest” machines which follow a given protocol. We also consider adversary classes with setup, in which case each adversary in \( Q \) is of the form \( QT^\ast \), where \( T \) is a program in a setup class \( \mathcal{T} \).

**3.2 Reach Extractor**

To define a reach extractor, we introduce some notation. We write \( QT \mathcal{E} \) to denote a composite machine in which \( \mathcal{E} \) semi-honestly runs \( Q \) internally in a straight-line manner (where \( \mathcal{E} \) can read the internal state of \( Q \)), letting \( Q \) directly communicate externally (with a generator). \( \mathcal{E} \) produces the final auxiliary output. For an adversary class with a setup, given an adversary \( QT^\ast \mathcal{E} \) in the composite machine \( QT \mathcal{E} \), \( \mathcal{E} \) is allowed to replace \( T \) with any program from \( \mathcal{T} \). For example, to capture common reference strings as a setup, \( \mathcal{T} \) would correspond to \{Setup, SetupSim\}, where Setup is the standard setup algorithm and SetupSim produces a simulated CRS.

As discussed in Section 2.2, \( \mathcal{E} \) is a valid reach-extractor if the following hold: in the IDEAL interaction, \( \mathcal{E} \) observes the adversary \( Q \) and produces an extra output \((\Pi, \delta^\ast)\) such that the states reached in \( \Pi \) using
$X^*$ (that is, $\text{REACH}_\Pi(X^*)$) is an upper bound on what $D$ can reach in $\pi^{(\alpha)}$; further the output is such that it reaches at-most a single state in each partition.\(^9\)

**Definition 3 (Reach-Extractor for $Q$ w.r.t. $(\hat{G}, \hat{\mathcal{P}})$).** A reach-extractor for an adversary $Q \in \mathcal{Q}$ w.r.t. a $(\mathcal{P}, \mathcal{M})$-generator class $\mathcal{G}$ and a transition function class $\mathcal{P}$, is a PPT program $\mathcal{E}$ such that, for all $G \in \mathcal{G}$ and PPT $D$, the output $X$ produced by the following two experiments are indistinguishable:

\[
\text{REAL}(G, Q, D): \quad \left( (\pi^{(\alpha)}, \mu^{(\beta)}), a_G, a_Q \right) \leftarrow \langle G : Q \rangle \\
\text{output } X \leftarrow D(\pi^{(\alpha)}, \mu^{(\beta)}, a_G, a_Q)
\]

\[
\text{IDEAL}(G, Q, \mathcal{E}, D): \quad \left( (\pi^{(\alpha)}, \mu^{(\beta)}), a_G, a_Q, X^* \right) \leftarrow \langle G : Q \rangle \mathcal{E} \\
\text{output } X \leftarrow D(\pi^{(\alpha)}, \mu^{(\beta)}, a_G, a_Q)
\]

and further the following hold:

- In $\text{IDEAL}(G, Q, \mathcal{E}, D)$, $\Pi \in \hat{\mathcal{P}}$.
- Suppose $\mathcal{I}$ partitions $\Sigma$ into $N$ parts $\Sigma_1, \cdots, \Sigma_N$. Then
  - **Reach-Bound**: For all $i \in [N]$, $\Pr[ (\text{REACH}_{\pi^{(\alpha)}}(X) \cap \Sigma_i) \not\subseteq (\text{REACH}_\Pi(X^*) \cap \Sigma_i) ]$ is negligible.
  - **Reach-Restriction**: For all $i \in [N]$, $|\text{REACH}_\Pi(X^*) \cap \Sigma_i| \leq 1$. \hfill \qed

**Real and Ideal Program Classes.** Note that, in the above definition, $\mathcal{E}$ in the ideal world is allowed to extract an idealized reactive program $\Pi \in \mathcal{P}$ to describe the set of states reachable by the adversary $Q$. While in many of our examples, $\hat{\mathcal{P}}$ is the same as $\mathcal{P}$, the class of “real world” reactive programs being obfuscated, this is not mandatory. This flexibility in the ideal world can help with enabling reach extraction while remaining useful in a higher level application. For example, looking ahead, in the case of commitment opening (Section 4.1), the reactive program in the real world transitions out of the start state on an input that is a valid opening of a commitment hardwired into the program, while the one in the ideal world requires the input to simply be the message that this commitment can be opened to. This is to accommodate the construction (based on a UC-secure OT) in which the reach extractor (using the simulator for the OT) can extract only the message in the commitment and not necessarily the opening of the commitment (randomness used in the OT protocol). This still suffices for applications like 2-round MPC, where either the commitments are created by the adversary, or the commitments in the reachable states are meant to be revealed to the adversary.

### 3.3 Reach-Restricted Reactive Program Obfuscation

Recall that, our goal in obfuscating a reactive program is to hide the parameters $\alpha, \beta$, except for the states an adversary can reach. Let $\mathcal{E}$ be a reach-extractor for $Q$ w.r.t. $\hat{\mathcal{G}}$ s.t. $\mathcal{E}$ outputs $\Pi, X^*$ and $\text{REACH}_\Pi(X^*)$ bounds the reach of the adversary in $\pi^{(\alpha)}$. Then, we define a secure obfuscation as requiring a simulator $\text{Sim}$ which, given only the circuits $\pi^{(1)}, \mu^{(1)}$ and the reachable states $(\overline{\pi}, \mu^{(\beta)}(\Pi(\overline{\pi})))$ for input sequences $\overline{\pi} \in X^*$, can output an obfuscation indistinguishable from a real obfuscation.

**Definition 4 (R3PO scheme $\mathcal{O}$ for $(G, Q, \mathcal{P})$).** A PPT program $\mathcal{O}$ is an **Reach-Restricted Reactive Program Obfuscation (R3PO) scheme** for a $(\mathcal{P}, \mathcal{M})$-Generator class $\mathcal{G}$ and transition function class $\mathcal{P}$, if the following hold:\(^10\)

\[^9\] As discussed just before Section 2.3, the reach-restriction condition is to enable our composition theorem (which depends on the use of garbled circuits).

\[^10\] We assume that the programs $\pi^{(\alpha)}, \mu^{(\beta)}, \mathcal{I}, \mathcal{O}$ are all specified as circuits. Further, $\pi^{(\alpha)}$ and $\mu^{(\beta)}$ are given as circuits for $\pi^{(1)}$ and $\mu^{(1)}$ (resp.), which take $\alpha$ and $\beta$ (resp.) as an input.
- **Correctness:** For all $\pi^{(\alpha)} \in \mathcal{P}$, $\mu^{(\beta)} \in \mathcal{M}$, $\rho \leftarrow \mathcal{O}(\pi^{(\alpha)}, \mu^{(\beta)})$, and $\pi \in X^*$, it holds that $\rho(\pi) = \mu^{(\beta)}(\pi^{(\alpha)}(\pi))$.

- **Security:** There exists a PPT program $\text{Sim}$ s.t. $\forall Q \in \mathcal{Q}$, there exists a reach-extractor $\mathcal{E}$ w.r.t. $(\mathcal{G}, \mathcal{P})$, so that $\forall G \in \mathcal{G}$, the outputs of the following two experiments are indistinguishable:

$$\left( (\pi^{(\alpha)}, \mu^{(\beta)}), a_G, a_Q \right) \leftarrow \langle G : Q \rangle$$

\[
\text{REAL}(G, Q): \\
\quad \rho \leftarrow \mathcal{O}(\pi^{(\alpha)}, \mu^{(\beta)}) \\
\quad \text{output } (\rho, a_G, a_Q)
\]

$$\left( (\pi^{(\alpha)}, \mu^{(\beta)}), a_G, \pi, X^* \right) \leftarrow \langle G : Q | \mathcal{E} \rangle$$

\[
\text{IDEAL}(G, Q): \\
\quad \rho \leftarrow \text{Sim}(\pi^{(\alpha)}, \mu^{(\beta)}, \pi, \{\pi, \mu^{(\beta)}(\pi(\pi))\}_{\pi \in X^*}) \\
\quad \text{output } (\rho, a_G, a_Q)
\]

\[\triangleq\]

### 4 A Library of One-Step Program Obfuscators

We now give examples of simple generators that output “one-step” reactive programs that have a single transition from the start state, and R3PO obfuscations for them w.r.t. some standard cryptographic primitives. The examples will also be useful as one-step program obfuscations to build obfuscations for bigger reactive programs (via our composition theorem, Theorem 1).

#### 4.1 Commitment Opening

Let $\text{Com} = (\text{Setup}, \text{Commit}, \text{Open})$ be a UC-secure extractable commitment scheme. We define a class of reactive programs, where the one-step transition corresponds to the opening of a commitment. The transition function $\pi_{c,\text{crs}}$ of such a reactive program is shown in Figure 1, with the associated partitioning $\Sigma = \Sigma_1 \cup \Sigma_2$.

![Transition function $\pi_{c,\text{crs}}$ is hardwired with a commitment $c$ and a setup $\text{crs}$. On input $d$, it transitions from start to $\sigma_m \in \Sigma_2$ iff $d$ is a valid opening of $c$ to $m$.](image)

Given a commitment scheme $\text{Com}$ and a message function class $\mathcal{M}$, we define the generator class $\mathcal{Q}^{\text{Com},\mathcal{M}}_{\text{COMM}}$ as consisting of generators which behave as follows: It accepts $(\text{crs}, c)$ from $\mathcal{Q}$, picks a message function $\mu^{(\beta)} \in \mathcal{M}$ arbitrarily, and outputs the reactive program $(\pi_{c,\text{crs}}, \mu^{(\beta)})$.

We define the adversary class $\mathcal{Q}^{\text{Com}}_{\text{COMM}}$ as a set of adversaries of the form $\mathcal{Q}^T$ for PPT $Q$ and setup $T \in \{\text{Setup}, \text{Setup}_\text{Sim} \}$. In the real experiment, $T$ will correspond to the real setup, while in the ideal experiment (of Definition 4), a reach extractor can replace it with a simulated setup that allows it to extract a message $m$ from $c$. The relaxed transition function that the reach extractor outputs, defined to be from a class $\mathcal{P}_{\text{COMM}}$ independent of the scheme $\text{Com}$, will only have a single state $\sigma_m$ reachable from start, and the transition happens on input $m$ itself (so that $X^*$ consists of just one input $m$). We remark that for any binding commitment $\text{Com}$, the reach restriction property will hold for this reach extractor, as an adversary can open $c$ to no other value than $m$.

\[\text{Using notation defined in Section 5.1, we may write } \mathcal{Q}^{\text{Com},\mathcal{M}}_{\text{COMM}} = \{ H \| Z \mid Z \text{ PPT} \}, \text{ where } H \text{ is a fixed machine and } Z \text{'s role is to specify the message function to } H.\]
The first set of additions to our R3PO library is stated below. It also includes a version for “weakly-secure” commitments which disallows corruption of the committing party during the commitment phase, and correspondingly an adversary class \( Q^{\text{Com}}_{\text{COMMIT}} \) with adversaries which construct \( c \) honestly as a commitment.

**Lemma 1.** Given a UC-secure 2-round OT scheme in the CRS model, there exists a UC-secure commitment scheme \( \text{Com} = \text{(Setup, Commit, Open)} \) and an R3PO scheme w.r.t. \( (G^\text{Com}, M, G^\text{Commit}, P^\text{COMMIT}) \) for any message function class \( \mathcal{M} \).

Further, given a semi-honest secure 2-round OT scheme, there exists a weakly-secure commitment scheme \( \text{Com} = \text{(Commit, Open)} \) and an R3PO scheme w.r.t. \( (G^\text{Com, M}, G^\text{Commit-Sh}, P^\text{COMMIT-Sh}) \) for any message function class \( \mathcal{M} \).

This construction, given in Appendix C.1, repurposes the OT scheme as a commitment scheme, following [7, 27]. 2-round OT itself can be based on a variety of standard assumptions such as DDH ([2, 51, 52]), quadratic residuosity assumption ([41, 52]) or learning with errors assumption ([52]). Note that the above result is stated for all \( \mathcal{M} \). Later, in Section 5.1 we will define a special message function class \( \tilde{\mathcal{M}} \), and it would be for \( \tilde{\mathcal{M}} \) that we will prove the above result, with the more general version following as a corollary of our composition theorem. In the sequel, \( G^\text{Com} \) will be abbreviated as \( G^\text{COMMIT} \).

### 4.2 Signature Checking

Let \( \text{Sig} \leftarrow \text{(gen, sign, verify, punct, psign)} \) be a puncturable signature scheme. We define a class of reactive programs, where the one-step transition corresponds to checking the signature of a fixed message on a fixed verification key. The transition function \( \pi_m \) of such a reactive program is shown in Figure 2, with its associated state space partition \( \Sigma = \Sigma_1 \cup \Sigma_2 \).

**Fig. 2.** Transition function \( \pi_{\text{mpre}}[\sigma_{\text{vk}}] \) has mpre hardwired into it, and a verification key \( \text{vk} \) encoded in the start state \( \sigma_{\text{vk}} \). On input \( (\tau, m) \), it transitions from the start state to \( \sigma_{\text{m}} \) if \( \tau \) is a valid signature on \( m \) w.r.t. \( \text{vk} \) and \( m \) has mpre as a prefix.

We define the generator class \( G^\text{Sig, M}_{\text{SIGN}} \) as consisting of generators which behave as follows: It samples a signature key pair \((\text{sk}, \text{vk})\), answers polynomially many signature queries for messages \( m_i \) from \( Q \) with \( \text{sign(}\text{sk}, m_i) \) and punctures \( \text{sk} \) at \( m_i \); then it receives a target message prefix mpre from \( Q \), sends the key \( \text{sk} \) punctured at mpre to \( Q \), picks a message function \( \mu(\beta) \in \mathcal{M} \) and outputs the program \( \langle \pi_{\text{mpre}}[\sigma_{\text{vk}}], \mu(\beta) \rangle \).

We define the adversary class \( Q^\text{Sig}_{\text{SIGN}} \) as set of all PPT \( Q \). The relaxed transition function that the reach extractor outputs is identical to to the transition function \( \pi_{\text{mpre}} \). A reach extractor can report the signed message received by \( Q \) in \( X^* \). While the reach restriction property trivially holds here (at most a single message with prefix mpre is signed), the reach bounding property relies on the unforgeability of \( \text{Sig} \). Also, by encoding \( \text{vk} \) in the start state, we force the “relaxed program” to be identical to that produced by the generator, and thereby prevent an arbitrary extractor from reaching \( \sigma_{\text{m}} \) unless \( Q \) received a signature on \( m \).

The following lemma adds signature checking to our R3PO library.

**Lemma 2.** If there exists a semi-honest secure OTSE scheme, then there exists a puncturable signature scheme \( \text{Sig} \) and an R3PO scheme \( O_{\text{Sig}} \) w.r.t. \( (G^\text{Sig, M}_{\text{SIGN}}, Q^\text{Sig}_{\text{SIGN}}, P^\text{SIGN}) \) for any message function space \( \mathcal{M} \).

We defer the proof to Appendix C.2.
4.3 Other Examples

As described in Section 2.4, we also incorporate R3POs for hash-opening and $\epsilon$-transition into our library. These are described in Appendix C. We also point out, in Appendix C.5, that we can adapt an R3PO so that the state space and partition function can be transformed in simple ways (e.g., by adding a fixed prefix to all the states in a part).

5 A Composition Theorem for R3PO

We now describe our composition theorem that enables building an R3PO for a generator class from R3POs for generator classes that produces smaller “one-step” (or non-reactive) programs. First we formalize the notion of decomposition.

5.1 Decomposition

The goal of decomposition is to view the transition function $\pi$ of a reactive program produced by a generator $G$, as consisting of several one-step transitions $\pi_i$ of reactive programs produced by generators $H_i$. Below, we define the notion of a $\sigma$-restriction of $\pi$ at a state $\sigma$.

One-Step Restriction of a Transition Function. Given a reactive program’s transition function $\pi^{(a)}$ and one of its states $\sigma$, we define a one-step $\sigma$-restriction of $\pi^{(a)}$ as a transition function $\pi^{(a)}_{\sigma}$ with start state $\sigma$, where

$$
\pi^{(a)}_{\sigma}(\sigma', x) = \begin{cases} 
\pi^{(a)}(\sigma, x) & \text{if } \sigma' = \sigma \\
\sigma & \text{otherwise.}
\end{cases}
$$

(i.e., in $\pi^{(a)}_{\sigma}$, the only transitions allowed are from its start state $\sigma$).

Note that the state space $\Sigma$ of $\pi$ can be exponentially large in $\kappa$, and correspondingly $\pi$ consists of that many one-step transition functions. When decomposing $\pi$, we will group them into polynomially many classes of transition functions, using the partition $I$ of the state space, $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_N$ associated with $\pi$. This imposes the following structure on the class of transition functions $\mathcal{P}$ to which $\pi$ belongs.

The transition function class $\mathcal{P}_1 \times \cdots \times \mathcal{P}_N$. For any set of $N$ classes $\mathcal{P}_1, \cdots, \mathcal{P}_N$ over the (same) state space $\Sigma$ and partition function $I : \Sigma \rightarrow [N]$, we define $\mathcal{P}_1 \times \cdots \times \mathcal{P}_N$ to consist of transition functions $\pi^{(a)}$ such that for each state $\sigma \in \Sigma_i$, the one-step $\sigma$-restriction of $\pi^{(a)}$ is in $\mathcal{P}_i$. That is, for all $\sigma \in \Sigma$ and inputs $x$, $\pi^{(a)}(\sigma, x) = \pi^{(a)}_{\sigma}(\sigma, x)$ where $\pi^{(a)}_{\sigma} \in \mathcal{P}_{I(\sigma)}$.

Though $\pi$ can have exponentially many states, we would like to view it as composed of $N$ transition functions, $\pi^{(a)}_{\sigma_i} \in \mathcal{P}_i$, where $\sigma_i \in \Sigma_i$. Thanks to the reach-restriction requirement on the reactive programs that we are interested in, for each $i$, there would indeed be only one state $\sigma_i \in \Sigma_i$ that we need to consider. However, recall that $\pi$ is dynamically generated by a generator $G$ interacting with an adversary $Q$, and the reachable states in $\pi$ are determined by this interaction. So the decomposition should be framed at the level of the interactive generators, rather than individual transition functions.

This leads us to a bi-simulation based definition of decomposition that views the generator $G$ as incorporating another generator $H$ (which produces one-step programs), and gives a two-way equivalence between them. To formalize this notion of bi-simulation, we introduce the following notation of composite machines.

Composite machines. It will be convenient to define a few different ways in which a program (a generator or an adversary) can be wrapped by another program. As described below, a generator will be wrapped by a blackbox simulator, $J$ or $Z$.

12 Looking ahead, the role of $J$ below is to simulate the presence of a one-step generator $H$ when the actual execution involves the generator $G$, and the role of $Z$ is to simulate the presence of $G$ when the actual execution involves $H$. 

12
- For a generator $H$, we write $\begin{array}{c} Z \\ H \end{array}$ to denote a composite machine in which $Z$ runs $H$ internally in a blackbox straight-line manner. The reactive program output by the composite generator is produced $H$, and the auxiliary output produced by it contains outputs from both $H$ and $Z$. $H$ may communicate with $Z$, and further the composite machine can communicate externally as described shortly. The running time of $\begin{array}{c} Z \\ H \end{array}$ is bounded by that of $H$ plus an additive $\text{poly}(\kappa)$ overhead that depends on $Z$.

- For a generator $G$, we write $\begin{array}{c} G \\ J \\ Z \end{array}$ to denote a slightly different composite machine, which is similar to $\begin{array}{c} J \\ G \end{array}$ ($G$ is run internally by $J$ in a blackbox straight-line manner, and the auxiliary information is produced by both) but the reactive program it produces is output by $J$. The external communication pattern is also different as described below.

- For an adversary $Q$, we write $\begin{array}{c} Z \\ H \end{array}$ to denote a composite machine in which $W$ internally runs $Q$ in a straight-line manner with additive overhead, but $W$ can read the internal state of $Q$. The auxiliary output of this composite machine is the entire view of $W$ (which includes the auxiliary information $a_Q$ produced by $Q$). The communication pattern is described below.

- Each of $\begin{array}{c} G \\ J \\ Z \end{array}$ and $\begin{array}{c} W \\ Z \end{array}$ has three external communication channels – one used by the internal machine (shown boxed) and the other two by the wrapper machine. In all machines the “middle” channel is used by the wrapper ($J$, $Z$ and $W$, respectively); the “top” channel is used by $G$, $Z$ and $Q$ (resp.); the “bottom” channel is used by $J$, $H$ and $W$ (resp.). Note that when $\begin{array}{c} G \\ J \\ Z \end{array}$ is connected to $\begin{array}{c} Z \\ W \end{array}$, $Q$ directly interacts with $G$, whereas when $\begin{array}{c} Z \\ W \end{array}$ is connected to $\begin{array}{c} Z \\ W \end{array}$, $Q$ interacts with $Z$. For the ease of writing expressions, we shall denote $\begin{array}{c} G \\ J \end{array}$ by $G[J]$, and $\begin{array}{c} Z \\ H \end{array}$ by $H[Z]$. We will denote $\begin{array}{c} Z \\ W \end{array}$ by $Q[W]$; in fact, we will be interested in $\begin{array}{c} Z \\ W \end{array}$ (where $Q[\mathcal{E}]$ itself is a composite machine involving a reach-extractor which interacts with $Q$ as defined in Definition 3); we shall denote it by $Q[\mathcal{E}]$.  

**Partial Reach-Extractor:** For a valid decomposition, it will be important to have a bi-simulation that maps $\pi$ to a one-step restriction $\pi_{\sigma}$ such that $\sigma$ is the unique reachable state in a subset of states $\Sigma_i$. To enforce this, we shall rely on an extractor for the adversary $Q$ w.r.t. the generator $G$ that produces $\pi$. However, the purpose of decomposition and composition is to be able to obtain an extractor for $Q$ w.r.t. $G$ along with a simulator, as in the definition of $R3PO$ (Definition 4). To break this apparent circularity, we use the notion of a partial reach extractor: An $(i - 1)$-partial reach extractor will be sufficient for defining decomposition “at part $\Sigma_i$,” and it can be extended to an $i$-partial reach extractor, using the $R3PO$ guarantee for the one-step generator.

Formally, a $t$-partial reach-extractor is defined identically to Definition 3, but with the relaxation that the reach-bound condition needs to hold only for $i \leq t$, instead of $i \leq N$. (The reach-restriction condition is still required to hold for all $i \in N$.) Thus, an $N$-partial reach extractor is a “full” reach-extractor.

Now we are ready to state the definition of decomposition. While informally we shall refer to decomposing a reactive program (or even a transition function) to one-step programs, formally, the decomposition is of a generator class to a sequence of generator classes, specified along with corresponding adversary classes and relaxed program classes.

**Definition 5 (Decomposition of $(G, Q)$ to $L$).** Let $G$ be a $(\mathcal{P}, \mathcal{M})$-generator class where $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_N$ is tree-ordered. Let $L = (\mathcal{H}, Q, \mathcal{P})$, where $\mathcal{H} = \{H[Z] \mid \text{PPT } Z\}$ for a fixed $(\mathcal{P}_i, \mathcal{M}_i)$-generator $H$, $Q$ is an adversary class, and $\mathcal{P}$ is a transition function class.
Then, a generator \( G \in \mathcal{G} \) is said to be \textit{decomposable at part} \( i \) \textit{to} \( L \) if, there exist PPT \( J, Z, W \) so that \( \forall Q \in \mathcal{Q} \), and all \((i - 1)\)-partial reach-extractors \( \mathcal{E} \) for \( Q \) w.r.t. \((\mathcal{G}, \mathcal{P})\), it holds that \( Q[\mathcal{E}]W \in Q \) and:

- \textbf{Indistinguishability:} \( \langle G \parallel J : Q[\mathcal{E}]W \rangle \approx \langle H \parallel Z : Q[\mathcal{E}]W \rangle \).
- \textbf{Correct One-Step Restriction:} \( \text{REACH} \Pi(X^*) \cap \Sigma_i \subseteq \{\sigma\} \).

\( \mathcal{G}, Q \) is said to be \textit{decomposable into} \( \mathcal{L} = (\mathcal{L}_1, \ldots, \mathcal{L}_N) \) if \( \forall G \in \mathcal{G}, \ i \in [N] \), it holds that \( \mathcal{L}_i = (\mathcal{H}_i, \mathcal{Q}_i, \mathcal{P}_i) \) where \( \mathcal{Q}_i \supseteq \mathcal{Q} \), and \( G \) is decomposable at part \( i \) to \( \mathcal{L}_i \).

An Illustrative Example. We illustrate the above decomposition definition via an example. The programs we decompose correspond to the “Signature + Commitment” example from Section 2.3. Recall that such a program makes two moves: (1) from a start state that includes a verification key \(vk\), it accepts an input \((c, s)\), and if \( s \) is verified as a signature on \( c \) w.r.t. \( vk\), then it moves to a state specifying the string \( c \); (2) then it accepts \((d, m)\), and if \( d \) is an opening of \( c \) as a commitment to \( m \), it moves to a state specifying \( m \).

The corresponding transition function family is \( \mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2 \), where \( \mathcal{P}_1 \) corresponds to signature verification and \( \mathcal{P}_2 \) to commitment opening, from Section 4. We consider a generator \( G \) which samples a signature key-pair \((sk, vk)\), uses \( sk \) to sign a single signature request, and outputs a program in \( \mathcal{P} \) above, with \( vk \) in its start state.

To illustrate that \( G \) decomposes at part 2 to \( G_{\text{COMMIT}} \) we use the following simulators:

- \( Q \) receives \( vk \) from \( G \) (or \( Z \)) and \( Q \) sends back a single signature request on a string \( c \) (if any). At this point, \( W \) will forward \( c \) as a commitment, through its bottom channel that is connected to \( H \) or \( J \).
- \( J \) accepts \( c \) from \( W \), and outputs a reactive program whose transition function is from \( \mathcal{P}_2 \) with the start state encoding \( c \) (and a uniformly random message function from \( \mathcal{M}_2 \), which can be arbitrary).
- \( Z \) samples a signature key-pair \((sk, vk)\) and sends it to \( Q \) (through the top channel). Then, when the first signature request arrives from \( Q \), it responds by signing it using \( sk \) (any further requests will be ignored).

Finally, \( Z \) outputs a reactive program with a transition function from \( \mathcal{P} \) and start state encoding \( vk \).\(^{13}\)

It can be seen that \( J, Z, W \) above meet all the requirements for decomposition. In particular, the unforgeability of the signature scheme used to define \( \mathcal{P}_1 \) enforces the requirement that the output of any 1-partial reach extractor takes the program produced by \( G \) to the same state (if any) as the start state \( \sigma \) of the one-step restriction produced by \( J \).

5.2 Composition Theorem

Above, decomposition related the transition functions in \( \mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_N \) to those in each \( \mathcal{P}_i \). Before stating our composition theorem, we need to specify the message function space \( \tilde{\mu}_i \) of these one-step programs as well.

As described in Section 2.3, \( \tilde{\mu}_i \) should release garbled circuit labels for the state at which it is evaluated. For our purposes, it will be helpful to consider a labeling function (denoted below as \( \tilde{\beta} \)) which takes the part

\(^{13}\) For simplicity, we omit the message function from this description. \( Z \) would sample it from the same distribution as \( G \).
**Parameters:** Program classes $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_N$ with state-space $\Sigma = \{0, 1\}^n$, and partition function $\mathcal{I} : \Sigma \to [N]$. A message function class $\mathcal{M}$.

**Given One-Step Obfuscators:** For each $i \in [N]$, $\mathcal{O}_i$ (taking inputs in $\mathcal{P}_i \times \tilde{\mathcal{M}}_{\mathcal{I},i}$)

**Garbling Scheme:** Let $(\text{GCGarble}, \text{GCEval})$ be a garbling scheme.

**Obfuscator $\mathcal{O}$:**
- **Input:** $(\pi^{(\alpha)}, \mu^{(\beta)})$, where $\pi^{(\alpha)} \in \mathcal{P}$, $\mu^{(\beta)} \in \mathcal{M}$.
- Uniformly randomly sample $\hat{\beta} : [N] \times [n] \times \{0, 1\} \to \{0, 1\}^n$
- For each $i \in [N]$,
  - Define $\hat{\mu}_i$ to be encode$^{\mathcal{I},t}_{\hat{\beta}}$.
  - Define the function $f_i$ as follows (with a fresh random tape hard-coded for $\mathcal{O}_i$ and, if needed, $\mu^{(\beta)}$):
    $$f_i(\sigma) = \left( \mathcal{O}_i(\hat{\pi}_{\sigma}^{(\alpha)}, \hat{\mu}_i), \mu^{(\beta)}(\sigma) \right)$$
- Let $\text{GC}_i \leftarrow \text{GCGarble}(f_i, \hat{\beta}_i)$, where $\hat{\beta}_i(\ell, b) = \hat{\beta}(i, \ell, b)$ for $\ell \in [n], b \in \{0, 1\}$.
- Let $i_0 = \mathcal{I}(\text{start})$, where start is the start-state of $\pi^{(\alpha)}$.
Output $\left( f_{i_0}(\text{start}), \{\text{GC}_i\}_{i \in [N] \setminus \{i_0\}} \right)$, along with a “driver program.”

![Fig. 3. Obfuscator $\mathcal{O}$ used to prove Theorem 1.](image)

index $i$ as an input, along with a bit position $j$ and bit value $b$. Then $\hat{\mu}_i$ will be of the form encode$^{\mathcal{I},t}_{\hat{\beta}}$ defined below, which only retains the part of $\hat{\beta}$ for parts $i > t$.

**Definition 6 (Message function space $\tilde{\mathcal{M}}$).** Let $\Sigma = \{0, 1\}^n$, with a partition function $\mathcal{I} : \Sigma \to [N]$, and $\hat{\beta} : [N] \times [n] \times \{0, 1\} \to \{0, 1\}^n$. A state labeling function encode$^{\mathcal{I},t}_{\hat{\beta}} : \Sigma \to \{0, 1\}^n$ is defined as

$$\text{encode}^{\mathcal{I},t}_{\hat{\beta}}(\sigma) = \begin{cases} (\hat{\beta}(\mathcal{I}(\sigma), 1, a_1), \ldots, \hat{\beta}(\mathcal{I}(\sigma), n, a_n)) & \text{if } \mathcal{I}(\sigma) > t, \\ \perp & \text{otherwise}, \end{cases}$$

where $\sigma = (a_1, \ldots, a_n)$. Then, we define $\tilde{\mathcal{M}} = \bigcup_{\mathcal{I} : \Sigma \to [N], t \in [N]} \tilde{\mathcal{M}}_{\mathcal{I},t}$, where

$$\tilde{\mathcal{M}}_{\mathcal{I},t} = \left\{ \left( \text{encode}^{\mathcal{I},t}_{\hat{\beta}} \right) | \hat{\beta} : [N] \times [n] \times \{0, 1\} \to \{0, 1\}^n \right\} \quad \triangleq$$

We are now ready to state our composition theorem.

**Theorem 1.** Suppose $\mathcal{G}$ is a $(\mathcal{P}, \mathcal{M})$-generator class that is decomposable into $\mathcal{L} = \{\mathcal{H}_i, \mathcal{Q}_i, \mathcal{P}_i\}_{i \in [N]}$, such that, for each $i \in [N]$, $\mathcal{H}_i$ is a $(\mathcal{P}_i, \tilde{\mathcal{M}}_{\mathcal{I},i})$-generator class and there exists an R3PO scheme $\mathcal{O}_i$ for $(\mathcal{H}_i, \mathcal{Q}_i, \mathcal{P}_i)$. Then there exists an R3PO scheme $\mathcal{O}$ for $(\mathcal{G}, \mathcal{Q}, \mathcal{P})$ where $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_N$.

The obfuscator $\mathcal{O}$ used to prove Theorem 1 is shown in Figure 3. Please refer to Appendix B for the proof that it is an R3PO scheme.
6 Identity Based Functional Encryption (IBFE)

As an illustration of the use of R3PO, we present an alternative derivation of the IBE construction in [21, 22] using an R3PO for signatures (Section 4.2). In fact, our construction readily extends to identity-based functional encryption, which may be of further interest.

6.1 Definition for IBFE

Definition 7 (Identity Based Functional Encryption (IBFE)). An IBFE scheme for a functionality $F$ defined over $(K, M)$ consists of the following PPT algorithms:

- Setup($1^\kappa$) $\rightarrow$ (mpk, msk): On input security parameter $\kappa$, outputs a key-pair (mpk, msk).
- KeyGen(msk, id, $f$) $\rightarrow$ sk$_{id,f}$: On input master secret key msk, identity id and key $f \in K$, outputs a function key sk$_{id,f}$.
- Encrypt(mpk, id, $m$) $\rightarrow$ $c$: On input master public key mpk, identity id and message $m \in M$, outputs a ciphertext $c$.
- Decrypt(sk$_{id,f}$, $c$) $\rightarrow$ $y$: On input a function key sk$_{id,f}$ and ciphertext $c$, outputs a message $y$.

The following properties are required from the above algorithms.

- Completeness: For all security parameters $\kappa$, $\forall id \in \{0, 1\}^n$, $\forall f \in K$, $\forall m \in M$ it holds that, if $(mpk, msk) \leftarrow$ Setup($1^\kappa$), sk$_{id,f} \leftarrow$ KeyGen(msk, id, $f$), then:
  $\Pr[\text{Decrypt}(sk_{id,f}, \text{Encrypt}(mpk, id, m)) = F(f, m)] = 1$

- Security: For any PPT adversary $A = (A_1, A_2)$, there exists a negligible function negl(.) such that the following holds:
  $\Pr[\text{IND}_{\text{IBFE}}(A) = 1] \leq \frac{1}{2} + \text{negl}(\kappa)$

where IND$_{\text{IBFE}}$ is shown in Figure 4.

6.2 Construction for IBFE

In this section, we give a scheme for IBFE from the following primitives.

- a puncturable signature scheme (Definition 18) Sig, with message space $M = \{0, 1\}^{m+o+1}$, where $m$ is the length of identity strings and $o$ is the length of function keys in IBFE.
- an R3PO scheme $O_{\text{SIGN}}$ w.r.t. $(\mathcal{G}_{\text{SIGN}}, \mathcal{Q}_{\text{SIGN}}, \mathcal{P}_{\text{SIGN}})$ (defined in Section 4.2).

Lemma 3. If there exists a puncturable signature scheme Sig and an R3PO scheme w.r.t. $(\mathcal{G}_{\text{SIGN}}, \mathcal{Q}_{\text{SIGN}}, \mathcal{P}_{\text{SIGN}})$, then there exists a secure IBFE scheme.

Proof sketch: We show that the scheme in Figure 5 is a secure IBFE scheme.

We prove security via a reduction to the security of the R3PO scheme $O_{\text{SIGN}}$. Let Adv = $(A_1, A_2)$ be an adversary with advantage $\alpha$ in the IBFE security game. We construct a generator $G \in \mathcal{G}_{\text{SIGN}}$ that simply plays the role of the experiment in the IBFE security game; construct an adversary $Q$ that internally runs $A_1$; and construct a distinguisher $D'$ that internally runs $A_2$. Now, in the ideal execution, $D'$ and thus $A_2$ get a simulated obfuscation $\tilde{\rho}$ that is independent of the challenge bit $b$ and messages $m_0, m_1$ (beyond the output revealed). But, the real execution corresponds to the IBFE game and thus $D'$ also has advantage $\alpha$. Please refer to Appendix D for the full details of the proof.
Experiment IND\textsuperscript{IBFE}

**Parameter:** Let $\kappa$ be the security parameter. Let $\mathcal{F}$ be the functionality.

\begin{itemize}
  \item $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\kappa)$.
  \item $(s_{A_1}, \text{id}^*, m_0, m_1) \leftarrow A_1^{\text{KeyGen}(\text{msk}, \cdot)}(\text{mpk})$.
  \item $b \leftarrow \{0,1\}$.
  \item $c \leftarrow \text{Encrypt}(\text{mpk}, \text{id}^*, m_0)$.
  \item $b' \leftarrow A_2^{\text{KeyGen}(\text{msk}, \cdot)}(s_{A_1}, \text{mpk}, c)$.
  \item If $A_1$ or $A_2$ queried key for an $(\text{id}^*, f^*)$ such that $\mathcal{F}(f^*, m_0) \neq \mathcal{F}(f^*, m_1)$ or queried $\text{id}^*$ for more than one key, output a random bit $b \leftarrow \{0,1\}$.
  \item Else, output 1 if $b' = b$ else 0.
\end{itemize}

Fig. 4. IBFE Security Experiment.

\begin{itemize}
  \item Let $\kappa$ be the security parameter.
  \item Let $\text{Sig} = (\text{Sig}.\text{gen}, \text{Sig}.\text{sign}, \text{Sig}.\text{verify}, \text{Sig}.\text{punct}, \text{Sig}.\text{psign})$ be a puncturable signature scheme.
  \item Let $\mathcal{O}_{\text{SIGN}}$ be an R3PO scheme w.r.t. $(\mathcal{O}_{\text{SIGN}}^\text{Sig}, \mathcal{O}_{\text{SIGN}}^\text{Sig}, \mathcal{O}_{\text{SIGN}})$.

### IBFE scheme:

- **Setup($1^n$):**
  \[(\text{Sig}.\text{vk}, \text{Sig}.\text{sk}) \leftarrow \text{Sig}.\text{gen}(1^n)\]
  output $(\text{mpk}, \text{msk}) := (\text{Sig}.\text{vk}, \text{Sig}.\text{sk})$.

- **KeyGen($\text{msk}, \text{id}, f$):**
  parse $\text{msk}$ as $\text{Sig}.\text{sk}$
  $\tau_{\text{id}\|f} \leftarrow \text{Sig}.\text{sign}(\text{Sig}.\text{sk}, \text{id}\|f)$
  output $\text{sk}_{\text{id}, f} := (\tau_{\text{id}\|f}, \text{id}\|f)$.

- **Encrypt($\text{mpk}, \text{id}, m$):**
  parse $\text{mpk}$ as $\text{Sig}.\text{vk}$
  let $\mu_{\text{id}, \mathcal{F}}$ be as follows:
  \[
  \mu_{\text{id}, \mathcal{F}}^{(m)}(\sigma^*_i) = \begin{cases} 
  \mathcal{F}(f, m) & \text{if } i = 2 \text{ and } x = \text{id}\|f, \\
  \bot & \text{otherwise.}
  \end{cases}
  \]
  output $\mathcal{O}_{\text{SIGN}}^\text{Sig}(\tau_{\text{id}}^{(\cdot)}[\sigma^*_i_{\text{Sig}.\text{sk}}], \mu_{\text{id}, \mathcal{F}}^{(m)}[\mathcal{F}_{\text{SIGN}}])$

- **Decrypt($\text{sk}_{\text{id}, f}, \text{ct}$):**
  parse $\text{sk}_{\text{id}, f}$ as $(\tau, x)$
  parse $\text{ct}$ as $\rho$
  output $m \leftarrow \rho((\tau, x))$
\end{itemize}

Fig. 5. IBFE from R3PO for Signatures

### 7 Private Multi-Authority ABE

In this section, we define Private Multi-Authority ABE and show how to instantiate it from any CP-ABE scheme, using R3PO schemes for commitment-opening and signatures together. Section 2.6 gives an overview of the notion and the construction described below.
7.1 Definition for Private Multi-Authority ABE

Let the authorities in the system be \( A_1, \ldots, A_N \), s.t. each authority \( A_i \) publishes its public key \( mpk_i \) after a local non-interactive setup. Each authority \( A_i \) also has an attribute-granting policy \( \Theta_i^{gid} \) w.r.t. each gid. A sender encrypts a message \( m \) with a ciphertext-policy \( \phi \) under the public keys of all the authorities, s.t. a receiver with global identifier gid and attribute vector \( \bar{x} \) can decrypt it only if it has attribute key for \( \bar{x} \) and each authorities’ attribute-granting policies accepts (that is, \( \forall i \in [N], \Theta_i^{gid} (\bar{x}) = 1 \)) and the ciphertext-policy accepts (that is, \( \phi(\bar{x}) = 1 \)). To get the attribute key for attribute vector \( \bar{x} \), the receiver sends a key-request \( req_i \) to each authority \( A_i, \) gets back a key-component \( sk_{req_i} \), and combines all the key-components to construct the key \( sk_\bar{x} \).

We define security w.r.t. a corruption model where the adversary is allowed to maliciously corrupt the receiver and semi-honestly corrupt any subset of the authorities. If the receiver is honest, we require that any key-request \( req \) reveals nothing about \( \bar{x} \) to the adversary (even if it semi-honestly corrupts all the authorities). If the receiver is corrupt, we require that the adversary is unable to distinguish between encryptions of any \( m_0 \) and \( m_1 \) w.r.t. a policy \( \phi \), if it did not send a key-request for any \( \bar{x} \) that satisfies \( \phi \) to the honest authorities.

**Definition 8 (Private Multi-Authority ABE (p-MA-ABE)).** A p-MA-ABE scheme for \( N \) authorities, message space \( M \), class \( C \) of ciphertext-policies and class \( \Theta \) of attribute-granting policies, both over \( n \)-bit attributes, and global identifiers space GID consists of PPT algorithms as follows:

- **SetupAuth** (\( 1^\kappa \) ) \( \rightarrow \) (\( mpk, msk \)): On input the security parameter \( \kappa \), outputs the master keys for an individual authority.
- **Encrypt** (\( \{ mpk_i \}_{i \in [N]}, \phi, m \) ) \( \rightarrow \) ct: On input the master public keys of all authorities, a policy \( \phi : \{0, 1\}^n \rightarrow \{0, 1\} \) in \( C \) and a message \( m \in M \), outputs a ciphertext ct.
- **KeyRequest** (\( \{ mpk_i \}_{i \in [N]}, \text{gid, } \bar{x} \) ) \( \rightarrow \) (\( st, \{ req_i \}_{i \in [N]} \)): On input the master public keys of all authorities, a global identity gid \( \in \text{GID} \) and an attribute vector \( \bar{x} \in \{0, 1\}^n \), outputs a recipient state \( st \) and requests \( \{ req_1, \ldots, req_N \} \).
- **KeyGen** \( (i, msk_i, \Theta_i^{gid}, req_i, \{ mpk_j \}_{j \in [N]} ) \rightarrow sk_{req_i} \): On input an authority index \( i \in [N] \), master secret key \( msk_i \), an attribute-granting policy \( \Theta_i^{gid} \), a request \( req_i \) and the master public keys of all authorities, outputs a key component \( sk_{req_i} \) or \( \bot \).
- **KeyCombine** (\( st, \{ sk_{req_i} \}_{i \in [N]} \) ) \( \rightarrow \) sk\( _\bar{x} \): On input a recipient state \( st \) and a set of key components \( \{ sk_{req_i} \}_{i \in [N]} \), outputs a secret key \( sk_\bar{x} \).
- **Decrypt** (\( sk_\bar{x}, \text{ct} \) ) \( \rightarrow \) m: On input a secret key \( sk_\bar{x} \) and a ciphertext ct, outputs a message \( m \) or \( \bot \).

The following correctness and security properties are required:

1. **Correctness:** \( \forall \) security parameter \( \kappa \), number of authorities \( N \in \mathbb{N} \), identities gid \( \in \text{GID} \), messages \( m \in M \), ciphertext policies \( \phi \in C \), attribute granting policies \( \{ \Theta_i^{gid} \}_{i \in [N]} \), and attribute vectors \( \bar{x} \) s.t. \( \phi(\bar{x}) = 1 \) and \( \Theta_i^{gid}(\bar{x}) = 1 \) for all \( i \in [N] \), it holds that if:

\[
\forall i \in [N], \quad (mpk_i, msk_i) \leftarrow \text{SetupAuth}(1^\kappa)
\]
\[
(st, \{ req_i \}_{i \in [N]} ) \leftarrow \text{KeyRequest}(\{mpk_i \}_{i \in [N]}, \text{gid, } \bar{x})
\]
\[
\forall i \in [N], \quad sk_{req_i} \leftarrow \text{KeyGen} (i, msk_i, \Theta_i^{gid}, req_i, \{ mpk_j \}_{j \in [N]} )
\]
\[
sk_\bar{x} \leftarrow \text{KeyCombine} (st, \{ sk_{req_i} \}_{i \in [N]})
\]

then \( \Pr [\text{Decrypt} (sk_\bar{x}, \text{Encrypt}(\{ mpk_i \}_{i \in [N]}, \phi, m) ) = m ] = 1 \).
2. Security of encryption: For any PPT adversary $A = (A_0, A_1, A_2, A_3)$ with semi-honest corruption of any subset of authorities and malicious corruption of the receiver, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds in the experiment $\text{IND}_{\text{mesg}}^{p-\text{MA-ABE}}$ shown in Figure 6:

$$\Pr[\text{IND}_{\text{mesg}}^{p-\text{MA-ABE}}(A) = 0] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Parameter: Let $\kappa$ be the security parameter, $N$ be the number of authorities.

<table>
<thead>
<tr>
<th>Experiment $\text{IND}_{\text{mesg}}^{p-\text{MA-ABE}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>- $(s_0, H) \leftarrow A_0(1^\kappa)$.</td>
</tr>
<tr>
<td>- $({\mathsf{mpk}_i, \mathsf{msk}<em>i} \leftarrow \text{SetupAuth}(1^\kappa))</em>{i \in H}$.</td>
</tr>
<tr>
<td>- $(s_1, {\mathsf{mpk}<em>i}</em>{i \in [N] \setminus H}) \leftarrow A_1(s_0, {\mathsf{mpk}<em>i}</em>{i \in H})$.</td>
</tr>
<tr>
<td>Let $O_k(y, {\Theta_{\text{id}}^{\text{pub}}}_{i \in H}) := \text{KeyGen}(k, \mathsf{msk}<em>k, \Theta</em>{\text{id}}^{\text{pub}}, y, {\mathsf{mpk}<em>i}</em>{i \in [N]})$, where $y = (\text{gid}, c)$.</td>
</tr>
<tr>
<td>- $(s_2, \phi, m_0, m_1) \leftarrow A_2^{\text{Ok}(...)}(s_1)$.</td>
</tr>
<tr>
<td>- $b \leftarrow {0, 1}$.</td>
</tr>
<tr>
<td>- $ct \leftarrow \text{Encrypt}({\mathsf{mpk}<em>i}</em>{i \in [N]}, \phi, m_b)$.</td>
</tr>
<tr>
<td>- $b' \leftarrow A_3^{\text{Ok}(...)}(s_2, ct)$.</td>
</tr>
<tr>
<td>- If $A_2$ or $A_3$ queried any oracle with different key-policies ${\Theta_{\text{id}}^{\text{pub}}}_{i \in H}$ for the same global id $\text{gid}$, output $a \leftarrow {0, 1}$.</td>
</tr>
<tr>
<td>If $\exists \text{gid}, \bar{x}$ s.t. $\phi(\bar{x}) = 1$ and, $\forall k \in H$, $\Theta^{\text{pub}}<em>k(\bar{x}) = 1$, and $\exists r, k^* \in H$ s.t. $A_2$ or $A_3$ queried the oracle $O</em>{k^<em>}$ with req$_{k^</em>}$, where $(st, {\text{req}<em>k}</em>{k \in [N]}) \leftarrow \text{KeyRequest}(({\mathsf{mpk}<em>k}</em>{k \in [N]}, \text{gid}, \bar{x}; r)$, then output $a \leftarrow {0, 1}$.</td>
</tr>
<tr>
<td>Else, output $a = b \oplus b'$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment $\text{IND}_{\text{attr}}^{p-\text{MA-ABE}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>- $(s_0, {\mathsf{mpk}<em>i}</em>{i \in [N]}, \bar{s}_0, \bar{x}_1, \text{gid}) \leftarrow A_0(1^\kappa)$.</td>
</tr>
<tr>
<td>- $b \leftarrow {0, 1}$.</td>
</tr>
<tr>
<td>- $b' \leftarrow A_1(s_0, \text{KeyRequest}(({\mathsf{mpk}<em>i}</em>{i \in [N]}, \text{gid}, \bar{s}_0))$.</td>
</tr>
<tr>
<td>- Output $b \oplus b'$.</td>
</tr>
</tbody>
</table>

Fig. 6. p-MA-ABE security experiments.

3. Receiver Privacy against Semi-Honest Adversary: For any PPT adversary $A = (A_0, A_1)$ that corrupts each authority in a semi-honest way, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds in the experiment $\text{IND}_{\text{attr}}^{p-\text{MA-ABE}}$ shown in Figure 6:

$$\Pr[\text{IND}_{\text{attr}}^{p-\text{MA-ABE}}(A) = 0] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

7.2 Construction for Private Multi-Authority ABE

In this section, we give a scheme for p-MA-ABE from the following primitives:

- a CP-ABE scheme (Definition 10)
- a non-interactive UC secure commitment scheme (Definition 11)
- a puncturable signature scheme (Definition 18)
an R3PO scheme $O^1_{p-MA-ABE}$ w.r.t. $(G^1_{p-MA-ABE}, Q_{p-MA-ABE})$ and $(G^2_{p-MA-ABE}, Q_{p-MA-ABE})$, which we describe in Section 7.3.

Let the number of authorities be $N$, space of access policies $C$ correspond exactly to the policies supported by the underlying single-authority CP-ABE scheme.

Completeness requirement of Commitment scheme: We will additionally require that, given a commitment setup $Com$ for any $c, d$ s.t. $Com.Open(Com.crs, c, d) = m$, it holds that $(m, c, d)$ lies in the support of the commit algorithm, that is: there exists $r$ s.t. $(c, d) \leftarrow Com.Commit(Com.crs, m; r)$. Any commitment scheme can be enhanced to have this property, by modifying it as follows.

$$\text{Commit}'(Com.crs, m; r) = \begin{cases} (c, d) & \text{if } r = 0^k || c || d \text{ and } Com.Open(Com.crs, c, d) = m \\ \text{Com.Commit}(Com.crs, m; r) & \text{otherwise.} \end{cases}$$

### Protocol for Private Multi-Authority ABE

**Parameter:** Let $\kappa$ be the security parameter, $N$ be the number of authorities.

**Primitives:**
- A CP-ABE scheme $ABE = (ABE.Setup, ABE.KeyGen, ABE.Encrypt, ABE.Decrypt)$
- A Commitment scheme $Com = (Com.Setup, Com.Commit, Com.Open)$
- An obfuscation scheme $O_{p-MA-ABE}$

- **p-MA-ABE.SetupAuth($1^\kappa$):**
  - Sample the common random string $Com.crs \leftarrow Com.Setup(1^\kappa)$.  
  - Sample the signature keys $(Sig.vk, Sig.sk) \leftarrow \text{Sig.gen}(1^\kappa)$.  
  - Sample the ABE keys $(ABE.mpk, ABE.msk) \leftarrow ABE.Setup(1^\kappa)$.  
  - Set $mpk := (Com.crs, Sig.vk, ABE.mpk)$ and $msk := (Sig.sk, ABE.msk)$.  
  - Output $(mpk, msk)$.

- **p-MA-ABE.Encrypt ($pk, \phi, m$):**
  - For all $i \in [N]$, parse $mpk_i$ as $(Com.crs_i, Sig.vk_i, ABE.mpk_i)$.  
  - Sample $s_1, \ldots, s_N$ s.t. $s_1 + \ldots + s_N = m$.  
  - For all $i \in [N]$, compute $ABE.ct_i \leftarrow ABE.Encrypt(ABE.mpk_i, \phi, s_i)$.  
  - Output $ct := \{ABE.ct_1, \ldots, ABE.ct_N\}$.

- **p-MA-ABE.Decrypt ($sk_\kappa, ct$):**
  - Parse $sk_\kappa$ as $\{sk^{ABE}_\kappa, \ldots, sk^{ABE}_N\}$.  
  - Parse $ct$ as $\{ABE.ct_1, \ldots, ABE.ct_N\}$.  
  - For all $i \in [N]$, set $s_i = ABE.Decrypt(ABE.mpk_i, sk^{ABE}_i, ABE.ct_i)$.  
  - Set $m := s_1 + \ldots + s_N$.  
  - Output $m$.

- **p-MA-ABE.KeyRequest($pk, gid, \bar{x}$):**
  - For all $i \in [N]$, parse $mpk_i$ as $(Com.crs_i, Sig.vk_i, ABE.mpk_i)$.  
  - compute $(c_i, d_i) \leftarrow Com.Commit(Com.crs_i, gid[\bar{x}])$  
  - and set $req_i := (gid, c_i)$.  
  - Set $st := (gid, \bar{x}, \{d_i\}_{i \in [N]})$.  
  - Output $(st, \{req_i\}_{i \in [N]})$.  

25
– p-MA-ABE.KeyGen: Computes
  • Parse $msk$ as $(\text{Sig}, \text{sk})$.
  • For all $i \in \{1, \ldots, N\}$, parse $mpk_i$ as $(\text{Com}, \text{sk})$.
  • Parse $req_i$ as $(\text{gid}, \text{c})$.
  • Fix $\pi_{pp}^{(a)} \in \mathcal{P}_{\text{pp}}$ where $pp = (t, \text{Com}, \text{c})$, $\alpha = \Theta^{\text{gid}}_t$, and start $= \sigma^{1}_{\text{pk}, \text{gid}}$.
  • Fix $\mu^{(b)} \in \mathcal{M}_{\text{pp}}$ where $\beta = (0, \text{Sig}, \text{ABE})$.
  • Compute $\text{sk}_{\text{req}} \leftarrow \mathcal{O}_{\text{pp}}(\pi_{pp}^{(a)}, \mu^{(b)})$.
  • Output $\text{sk}_{\text{req}}$.

– p-MA-ABE.KeyCombine: Computes
  • Parse $\text{st}$ as $(\text{gid}, \{d_i\})$.
  • For all $i \in \{1, \ldots, N\}$, parse $\text{sk}_{\text{req}}$ as $\rho_i$ and evaluate as $\tau_i \leftarrow \rho_i(d_i)$.
  • Finally, for all $t \in \{1, \ldots, N\}$, compute $\text{sk}_{\text{ABE}} \leftarrow \rho_i(\tau_1, \ldots, \tau_N)$.
  • Output $(\text{sk}_{\text{ABE}})_{i \in \{1, \ldots, N\}}$.

Fig. 7. A secure p-MA-ABE Protocol.

We now prove that the protocol in Figure 7 is in fact a secure p-MA-ABE scheme.

**Lemma 4.** If there exists a CP-ABE scheme, a non-interactive UC secure commitment scheme, a puncturable signature scheme, and an R3PO scheme $\mathcal{O}_{\text{pp}}$ w.r.t. $(\mathcal{G}_1, \mathcal{Q}_{\text{pp}})$, then there exists a secure p-MA-ABE scheme.

Please refer to Appendix E.1 for the full details of the proof.

### 7.3 Modeling the Key-Component as an R3PO

We define two generator classes that will be useful to prove security of the scheme in Figure 7. At a high level, both generators pick the keys for the honest authorities. The first generator allows us to replace the signing key $sk_i$ of an honest authority $A_i$ in each reactive-program obfuscation for a key-request $\text{gid} \parallel c$ with a signature $\tau$ on $\bar{x}$ s.t. the commitment $c$ can only be opened to $\bar{x}$ (even for adversarially generated commitments $c$). The second generator allows us to then replace the ABE master secret key $msk_i$ of each honest authority $A_i$ in the obfuscation with: either the ABE key for $\bar{x}$ (if the attribute-granting policies of all the honest authorities agree) or empty key.

**Program class:** Each program $(\pi_{pp}^{(a)}), \mu^{(b)})$ in $\mathcal{P}_{\text{pp}}$ is specified as follows:

- public parameter $pp = (t, \text{Com}, \text{c})$ (hardcoded in the program)
- start state $\sigma^{1}_{\text{pk}, \text{gid}}$, where $\text{pk} = \{\text{Sig}, \text{vk}_k\}_{k \in \{1, \ldots, N\}}$
- transition function $\pi_{pp}^{(a)}(\sigma^{1}_{\text{st}}, y)$, where $\alpha = \Theta^{\text{gid}}$, is defined as:

\[
\pi_{pp}^{(a)}(\sigma^{1}_{\text{st}}, y) = \begin{cases} 
\sigma^{2}_{\text{pk}, \text{gid}, \text{gid}'}, \bar{x} & \text{where } t = 1, \text{st} = (\text{pk}, \text{gid}), \\
\sigma^{3,0}_{\text{pk}, \text{gid}}, \bar{x} & \text{if } \text{Com} \cdot \text{Open}(\text{Com}, \text{c}, y) = \text{gid}' \parallel \bar{x}, \\
\sigma^{3,1}_{\text{pk}, \text{gid}}, \bar{x} & \text{where } t = 2, \text{st} = (\text{pk}, \text{gid}, \text{gid}' \parallel \bar{x}), \\
\sigma^{3,2,0}_{\text{pk}, \text{gid}}, \bar{x} & \text{if } \Theta^{\text{gid}}(\bar{x}) = 1, \text{gid}' = \text{gid}, \\
\sigma^{3,2,1}_{\text{pk}, \text{gid}}, \bar{x} & \text{where } t = (3, k - 1), k \in \{1, \ldots, N\}, \text{st} = (\text{pk}, \text{gid}, \bar{x}), y = (\tau_k, m_k), \\
\text{if } \text{Sig} \cdot \text{verify}(\text{Sig}, \text{vk}_k, \tau_k, m_k) = 1, m_k \neq \text{gid} \parallel \bar{x}.
\end{cases}
\]

\(^4\) formally, $\rho_i$ also takes as input a state $\sigma_1$ and also outputs a state $\sigma_2$. For brevity, we ignore mentioning it here.  
\(^5\) which is also an R3PO w.r.t. $(\mathcal{G}_1, \mathcal{Q}_{\text{pp}})$.
message function $\mu^{(\beta)} \in \mathcal{M}_{p-MA-ABE}$, with $\beta = (b, ss, ABE.msk)$, for $b \in \{0, 1\}$ and ss of the form:

$$ss = \begin{cases} \text{Sig.sk} & \text{if } b = 0, \\ \tau & \text{otherwise}. \end{cases}$$

is defined as:

$$\mu^{(\beta)}(\sigma) = \begin{cases} \text{Sig.sign}(ss, gid||\bar{x}) & \text{if } b = 0, \sigma = \sigma_{pk,gid,\bar{x}}^{3,0}, \\ ss & \text{if } b = 1, \sigma = \sigma_{pk,gid,\bar{x}}^{3,0}, \\ \text{ABE.KeyGen}(ABE.msk_t, \bar{x}) & \text{if } \sigma = \sigma_{pk,gid,\bar{x}}^{3,N}, \\ \bot & \text{otherwise.} \end{cases}$$

Partition function $\mathcal{I}_{p-MA-ABE} : \Sigma \rightarrow [3 + N]$ defined as: $\mathcal{I}_{p-MA-ABE}(\sigma_1^1) = 1$, $\mathcal{I}_{p-MA-ABE}(\sigma_1^2) = 2$ and $\mathcal{I}_{p-MA-ABE}(\sigma_1^3) = 3 + t$ for $t \in \{0, \ldots, N\}$, and some state information st.

**Generator Class $G^1_{p-MA-ABE}$** : This class contains a class of generators of the form $G^1_H$, parameterized by a set $H$ (denoting the set of honest authorities), that interacts with an arbitrary PPT $Q$ as follows:
- it receives master public keys $\{mpk\}_{i\in[N]}$ of the authorities from $Q$.
- it finally receives a target request of the form $(t, gid, c, \{G^1_i\}_{i\in H})$, where $t \in H$ and $(\text{Sig.sk}_i, ABE.msk_i)$.
- it halts with output $(\pi^{(o)}_{pp}, \mu^{(\beta)}) \in \mathcal{P}_{p-MA-ABE}$, where $pp = (t, \text{Com.crs}_t, c)$, $\alpha = \Theta^1_t$, start = $\sigma_{pk,gid}^1$ and $\beta = (0, \text{Sig.sk}_t, ABE.msk_t)$.

**Generator Class $G^2_{p-MA-ABE}$** : This class contains a class of generators of the form $G^2_H$, parameterized by a set $H$ (denoting the set of honest authorities), that interacts with an arbitrary PPT $Q$ as follows:
- it receives commitment setups $\{\text{Com.crs}_i\}_{i\in H}$, ABE key pairs $\{(ABE.mpk_i, ABE.msk_i)\}_{i\in H}$ from $Q$.
- it samples signature key pairs $\{\text{Sig.sk}_i\}_{i\in H}$ and sends $\{\text{Sig.sk}_i\}_{i\in H}$ to $Q$.
- it receives signature verification keys $\{\text{Sig.vk}_i\}_{i\in[N]\setminus H}$ of the corrupt authorities from $Q$.
- For $k \in \text{poly}(1^n)$, it receives query of the form $(t, gid, \bar{x}, \{\Theta^1_i\}_{i\in H})$, and sends $\{\tau_{i}\}_{i\in H}$, where $\forall i \in H$:

$$\tau_i = \begin{cases} \text{Sig.sign}(\text{Sig.sk}_i, gid||\bar{x}) & \text{if } \Theta^1_i(\bar{x}) = 1, \\ \bot & \text{otherwise.} \end{cases}$$

it also updates the signature key for authorities $i \in H$ as:

$$\text{Sig.sk}_i = \begin{cases} \text{Sig.sk}_i & \text{if } \Theta^1_i(\bar{x}) = 1, \\ \text{Sig.punct}(\text{Sig.sk}_i, gid||\bar{x}) & \text{otherwise.} \end{cases}$$

- it finally receives a target request of the form $(t, gid, c, \bar{x}, \tau, \{G^1_i\}_{i\in H})$.
- it halts with output $(\pi^{(o)}_{pp}, \mu^{(\beta)}) \in \mathcal{P}_{p-MA-ABE}$ and auxiliary output $a_G = \{(\text{Sig.vk}_i, \text{Sig.sk}_i)\}_{i\in H}$, where $pp = (t, \text{Com.crs}_t, c)$, $\alpha = \Theta^1_t$, start = $\sigma_{pk,gid}^1$ and $\beta = (1, \tau, ABE.msk_t)$.

**Adversary Class $Q_{p-MA-ABE}$** : Set of all adversaries of the form $Q^{\{T_i\}_{i\in H}}$, for PPT $Q$ and each $T_i \in \{\text{Com.Setup, Setup}_{\text{Sim}}\}$, for $i \in H$.

---

$^{16}$Note that puncturing of the signing key is crucial here, both to allow decomposition to signature-checking one-step program as well as in the proof.
The obfuscation scheme: We show that the same obfuscation scheme is a valid R3PO w.r.t. both generators described above. Further, it can be decomposed to commitment-opening (Section 4.1) and signature-checking (Section 4.2) R3PO in the library. Thus, we have:

**Lemma 5.** If there exist R3PO schemes w.r.t. \((G_{\text{Com}}^\text{COMMIT}, Q_{\text{Com}}^\text{COMMIT}, P_{\text{Com}}^\text{COMMIT})\) and \((G_{\text{Sig}}^\text{SIGN}, Q_{\text{Sig}}^\text{SIGN}, P_{\text{Sig}}^\text{SIGN})\), where \text{Com} is a non-interactive UC-secure commitment scheme and \text{Sig} is a puncturable signature scheme, then there exists a compiler \(O_{\text{p-MA-ABE}}\) which is an R3PO scheme w.r.t. \((G_{\text{p-MA-ABE}}^{1}, Q_{\text{p-MA-ABE}}^{1})\) and also an R3PO w.r.t. \((G_{\text{p-MA-ABE}}^{2}, Q_{\text{p-MA-ABE}}^{2})\).

We defer the details of the proof to Appendix E.2. In particular, \(O_{\text{p-MA-ABE}}\) will be defined using Theorem 1, and the decomposition of \((G_{\text{p-MA-ABE}}^{2}, Q_{\text{p-MA-ABE}}^{2})\) w.r.t. the corresponding libraries for commitment-opening and signature-checking.

**References**


Appendix

A Additional Preliminaries

Definition 9 (Garbled Circuits). A Garbling scheme is a pair of PPT algorithms \((\text{GCkt}, \text{GCEval})\) such that

- \(\text{GC} \leftarrow \text{GCGarble}(C, \widehat{\beta})\): \(\text{GCkt}\) takes as input a circuit \(C\), and a set of labels \(\widehat{\beta}(= \{\widehat{\beta}^w, b : b \in \{0, 1\}, w \in \text{inputwire}(C)\})\) containing two labels per input wire of the circuit \(C\). The procedure outputs a garbled circuit \(\text{GC}\).
- \(\text{ev} \leftarrow \text{GCEval}(\text{GC}, \{\widehat{\beta}^w, x^w\}^w_{w \in \text{inputwire}(C)})\): \(\text{GCEval}\) takes two inputs, a garbled circuit \(\text{GC}\) and a sequence of input labels and outputs a string \(\text{ev}\).

Correctness. For correctness, we require that for each input \(x\) to the circuit \(C\),

\[
\Pr[C(x) = \text{GCEval}(\text{GC}, \text{lab}_x)] = 1
\]

where \(\text{GC} \leftarrow \text{GCGarble}(1^\kappa, C, \widehat{\beta})\) and \(x^w\) corresponds to the value on input wire \(w\)(while evaluating \(C\)) and \(\text{lab}_x = \{\widehat{\beta}^w, x^w\}^w_{w \in \text{inputwire}(C)}\).

Security. For security, we require that

1. There exists a simulator \(\text{GSim}\) such that for any circuit \(C\), its input \(x\) and uniformly random \(\widehat{\beta}\), we have that

\[
\text{GC}, \text{lab}_x \approx \text{GSim}(\Phi(C), \text{lab}_x, y)
\]

where \(\text{GC} \leftarrow \text{GCGarble}(1^\kappa, C, \widehat{\beta})\), \(\text{lab} = \{\widehat{\beta}^w, x^w\}^w_{w \in \text{inputwire}(C)}\) \(y = C(x)\) and \(\Phi(C)\) denotes the topology of the circuit \(C\).

2. There exists a simulator \(\text{GCircSim}\) such that for any circuit \(C\) and uniformly random \(\widehat{\beta}\), we have that

\[
\text{GC} \approx \text{GCircSim}(\Phi(C))
\]

where \(\text{GC} \leftarrow \text{GCGarble}(C, \widehat{\beta})\). This requirement guarantees that a garbled circuit (without given input labels for evaluation) can be simulated just from the topology of the circuit \(C\).

\begin{flushright}
\(\triangleleft\)
\end{flushright}

Definition 10 (Ciphertext-Policy ABE (ABE)). An ABE scheme for a message space \(M\), attribute space \(\{0, 1\}^l\) and a circuit class \(C\) consists of PPT algorithms defined as follows:

- \(\text{Setup}(1^\kappa, 1^l) \rightarrow (\text{mpk}, \text{msk})\): On input the security parameter \(\kappa\), outputs the master public key \(\text{mpk}\) and master secret key \(\text{msk}\).
- \(\text{KeyGen}(\text{msk}, x) \rightarrow \text{sk}_x\): On input master secret key \(\text{msk}\) and an attribute string \(x\), outputs a secret key \(\text{sk}_x\).
- \(\text{Encrypt}(\text{mpk}, \phi, m) \rightarrow \text{ct}\): On input master public key \(\text{mpk}\), a policy \(\phi : \{0, 1\}^l \rightarrow \{0, 1\}\) in \(C\) and a message \(m \in M\), outputs a ciphertext \(\text{ct}\).
- \(\text{Decrypt}(\text{mpk}, \text{sk}_x, \text{ct}) \rightarrow m\): On input master public key \(\text{mpk}\), a secret key \(\text{sk}_x\) and ciphertext \(\text{ct}\), outputs message \(m\).

The following correctness and security properties are required:
1. **Correctness**: ∀κ, messages m ∈ M, policies φ ∈ C, attributes x ∈ {0, 1} such that φ(x) = 1, it holds that if (mpk, msk) ← Setup(1κ, 1l), skx ← KeyGen(msk, x), then
   \[ \Pr [\text{Decrypt}(mpk, skx, \text{Encrypt}(mpk, φ, m)) = m] = 1 - \text{negl}(κ) \]

2. **Selective Security**: For any PPT adversary A = (A₀, A₁, A₂), there exists a negligible function negl(.) such that the following holds
   \[ \Pr [\text{IND}_{\text{SEL}}^{\text{ABE}}(A) = 1] ≤ \frac{1}{2} + \text{negl}(κ) \]
   where INDₕₜₑₜₜₜ is shown in 8.

---

**Fig. 8.** ABE ind-security Experiment.

<table>
<thead>
<tr>
<th>Parameter: Let κ be the security parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>- (s_A₀, φ) ← A₀(1^n).</td>
</tr>
<tr>
<td>- (mpk, msk) ← Setup(1^n, 1^[l]).</td>
</tr>
<tr>
<td>- (m₀, m₁, s_A₁) ← A₁[KeyGen(msk, .)](s_A₀, mpk).</td>
</tr>
<tr>
<td>- b ← {0, 1}.</td>
</tr>
<tr>
<td>- m := m_b.</td>
</tr>
<tr>
<td>- ct ← Encrypt(mpk, φ, m).</td>
</tr>
<tr>
<td>- b' ← A₂[KeyGen(msk, .)](s_A₁, mpk, ct).</td>
</tr>
<tr>
<td>- If A₁ or A₂ asked for secret key of any x such that φ(x) = 1, output r ← {0, 1}.</td>
</tr>
<tr>
<td>Else, output 1 if b = b' else 0.</td>
</tr>
</tbody>
</table>

---

3. **Full Security**: For any PPT adversary A = (A₁, A₂), there exists a negligible function negl(.) such that the following holds
   \[ \Pr [\text{IND}_{\text{FULL}}^{\text{ABE}}(A) = 1] ≤ \frac{1}{2} + \text{negl}(κ) \]
   where INDₙₐₜₜₜ is shown in 9.

**Definition 11 (Commitment scheme).** A UC-secure commitment scheme Com over input space M = {0, 1} consists of the algorithms (Setup, Commit, Open) with the following syntax:

- Setup(1^n) → crs : takes the security parameter κ as input (in unary), and outputs the common reference string crs.
- Commit(crs, x) → (c, d) : takes as input a common reference string crs, a string x, and outputs a commitment c, decommitment d.
Parameter: Let \( \kappa \) be the security parameter.

- \((\text{mpk, msk}) \leftarrow \text{Setup}(1^\kappa, 1')\).
- \((s_{A_0}, \phi) \leftarrow A_0(1^\kappa)\).
- \((\phi, m_0, m_1, s_{A_1}) \leftarrow A_1^{\text{KeyGen}(\text{msk})}(s_{A_0}, \text{mpk})\).
- \(b \leftarrow \{0, 1\}\).
- \(m := m_b\).
- \(ct \leftarrow \text{Encrypt}(\text{mpk}, \phi, m)\).
- \(b' \leftarrow A_2^{\text{KeyGen}(\text{msk})}(s_{A_1}, \text{mpk}, ct)\).
- If \(A_1\) or \(A_2\) asked for secret key of any \(x^*\) s.t. \(\phi(x^*) = 1\), output \(r \leftarrow \{0, 1\}\).
  Else, output 1 if \(b = b'\) else 0.

Fig. 9. ABE full ind-security Experiment.

- \(\text{Open}(\text{crs}, c, d) \rightarrow x :\) takes as input a common reference string \(\text{crs}\), a commitment \(c\), a decommitment \(d\), and outputs a string \(x\).

and the following properties:

- **Hiding.** For all \(\kappa\), all strings \(x_0, x_1 \in \mathcal{X}\) and PPT adversaries \(A\), there exists a negligible function \(\text{negl}(\cdot)\) such that if \(\text{crs} \leftarrow \text{Setup}(1^\kappa), (c_0, d_0) \leftarrow \text{Commit}(\text{crs}, x_0), (c_1, d_1) \leftarrow \text{Commit}(\text{crs}, x_1)\), then it holds that
  \[
  \left| \Pr \left[ A(\text{crs}, c_0) = 1 \right] - \Pr \left[ A(\text{crs}, c_1) = 1 \right] \right| \leq \text{negl}(\kappa)
  \]

- **Binding.** There exists a PPT extractor \(E = (E_1, E_2)\), such that for all PPT adversaries \(A = (A_1, A_2)\), it holds that
  \[
  \left| \Pr \left[ \text{IND}_{A}^{\text{REAL}(1^\kappa)} = 1 \right] - \Pr \left[ \text{IND}_{A,E}^{\text{IDEAL}(1^\kappa)} = 1 \right] \right| \leq \text{negl}(\kappa)
  \]

where the experiments are defined as follows:

**Experiment \(\text{IND}_{\mathcal{A}}^{\text{REAL}(1^\kappa)}:***

\[
\begin{align*}
\text{crs} & \leftarrow \text{Setup}(1^\kappa) \\
(st_{A_1}, c, d) & \leftarrow A_1(\text{crs}) \\
\text{Output } A_2(st_{A_1}, \text{Open}(\text{crs}, c, d))
\end{align*}
\]

**Experiment \(\text{IND}_{A,E}^{\text{IDEAL}(1^\kappa)}:***

\[
\begin{align*}
\text{(crs, st}_{E_1}) & \leftarrow E_1(1^\kappa) \\
(st_{A_1}, c, d) & \leftarrow A_1(\text{crs}) \\
\text{Output } A_2(st_{A_1}, E_2(st_{E_1}, c))
\end{align*}
\]

- **Equivocal Commitment.** There exists PPT simulator \(\text{Sim}_{Eq}\) such that for any \(x \in \mathcal{X}\), it holds that
  \[
  \left\{ (\text{crs}, (c, d_x)) : (\text{crs}, c, \{d_y | y \in \mathcal{X}\}) \leftarrow \text{Sim}_{Eq}(1^\kappa) \right\} \approx \left\{ (\text{crs}, \text{Commit}(\text{crs}, x)) : \text{crs} \leftarrow \text{Setup}(1^\kappa) \right\}
  \]

**Definition 12 (Oblivious Transfer (OT)).** A UC-secure OT scheme consists of four algorithms \((\text{Setup}, \text{OT}_1, \text{OT}_2, \text{OT}_3)\) with the following syntax:
– Setup($1^\kappa$) → crs: takes as input the security parameter $1^\kappa$, and outputs the common random string crs.
– OT1(crs, b) → (ots1, ω): takes as input a common random string crs, bit b, and outputs a tuple (ots1, ω), where ots1 is a ciphertext and ω is a secret state.
– OT2(crs, ots1, (m0, m1)) → ots2: takes as input a common random string crs, a ciphertext ots1, messages m0, m1, and outputs a ciphertext ots2.
– OT3(crs, ots2, ω) → mb: takes as input a common random string crs, a ciphertext ots2, a secret state ω, and outputs a message mb.

and the following properties:

– **Correctness.** It holds for all $\kappa$, every bit $b \in \{0, 1\}$, every string $\omega$, every pair of messages $m_0, m_1$, that if crs ← Setup($1^\kappa$), ots1 ← OT1(crs, b) and ots2 ← OT2(crs, ots1, (m0, m1)), then
  \[
  \text{OT3(crs, ots2, ω) = m_b}
  \]

– **Receiver’s Security.** It holds for all $\kappa$ that
  \[
  \Pr[\text{IND}_{\text{REAL}}^{\kappa}(1^\kappa, K_0, K_1) = 1] - \Pr[\text{IND}_{\text{IDEAL}}^{\kappa}(1^\kappa, K_0, K_1) = 1] \leq \text{negl}(\kappa)
  \]

\[\begin{align*}
\text{Experiment IND}_{\text{REAL}}^{\kappa}(1^\kappa, K_0, K_1): \\
\crs &\leftarrow \text{Setup}(1^\kappa) \\
(st_{A_1}, ots_1) &\leftarrow A_1(\crs) \\
\text{ots}_2 &\leftarrow \text{OT}_2(\crs, ots_1, K_0, K_1) \\
\text{Output } A_2(st_{A_1}, ots_2)
\end{align*}\]

\[\begin{align*}
\text{Experiment IND}_{\text{IDEAL}}^{\kappa}(1^\kappa, K_0, K_1): \\
(\crs, \tau) &\leftarrow \mathcal{E}_1(1^\kappa) \\
(st_{A_1}, ots_1) &\leftarrow A_1(\crs) \\
\beta &\leftarrow \mathcal{E}_2(\tau, ots_1) \\
L_0 &:= K_\beta \text{ and } L_1 := K_\beta \\
\text{ots}_2 &\leftarrow \text{OT}_2(\crs, ots_1, L_0, L_1) \\
\text{Output } A_2(st_{A_1}, ots_2)
\end{align*}\]

– **Equivocal Receiver’s Security.** There exists a PPT simulator SimEq such that for any $b \in \{0, 1\}$, it holds that
  \[
  \Pr[(\crs, (ots_1, \omega_b)) : (\crs, ots_1, \omega_0, \omega_1) \leftarrow \text{SimEq}] \approx \Pr[(\crs, \text{OT}_1(\crs, b)) : \crs \leftarrow \text{Setup}(1^\kappa)]
  \]

**Definition 13 (Block-Openable CRH).** A block-openable collision-resistant hash scheme consists of the algorithms (crsGen, hash, openBlock, acceptBlock) with the following syntax:

– crs ← crsGen($1^\kappa$). It is a probabilistic algorithm that takes as input the security parameter $1^\kappa$ and outputs a common reference string crs.
– digest ← hash(crs, m). It is a deterministic algorithm that takes as input a common reference string crs and a message m, and outputs a digest.
– w ← openBlock(crs, m, i). It is a deterministic algorithm that takes as input a common reference string crs, a message m and a block index i ∈ [n] s.t. m = m₁∥ ...∥mₙ, and outputs a string w.
– y ← acceptBlock(crs, digest, i, w). It is a deterministic algorithm that takes as input a common reference string crs, a hash output digest, a block index i and a string w, and outputs a string y.

and the following properties:

– **Correctness:** For all m ∈ M, and all crs in the support of crsGen,

\[ \text{acceptBlock}(crs, \text{hash}(crs, m), i, \text{openBlock}(crs, m, i)) = m_i, \]

where i ∈ [n] is a block index and m = m₁∥ ...∥mₙ.
– **Collision Resistance:** For all PPT adversaries A, the following probability is negligible:

\[ \Pr_{\substack{crs \leftarrow \text{crsGen}(1^\kappa) \cr (m, i, w) \leftarrow A(crs)}} [\text{acceptBlock}(crs, \text{hash}(crs, m), i, w) \notin \{\bot, m_i\}] . \]

We would typically require hash and openBlock to produce an output that is much shorter than the messages being hashed. We also provide the definition of the laconic OT primitive [18] below.

**Definition 14 (Laconic OT).** A laconic OT (ℓOT) scheme syntactically consists of four algorithms crsGen, hash, Send and Receive.

– crs ← crsGen(1^κ). It takes as input the security parameter 1^κ and outputs a common reference string crs.
– (digest, D) ← hash(crs, D). It takes as input a common reference string crs and a database D ∈ {0, 1}^* and outputs a digest digest of the database and a state D.
– ct ← Send(crs, digest, L, m₀, m₁). It takes as input a common reference string crs, a digest digest, a database location L ∈ N and two messages m₀ and m₁ of length κ, and outputs a ciphertext ct.
– m ← Receive(D(crs, ct, L). This is a RAM algorithm with random read access to D. It takes as input a common reference string crs, a ciphertext ct, and a database location L ∈ N. It outputs a message m.

We have slightly changed the sender privacy definition from the original paper by allowing the adversary to read the crs before outputting its challenge messages. We require the following properties of an ℓOT scheme (crsGen, hash, Send, Receive).

– **Correctness:** We require that it holds for any database D of size at most m = poly(κ), for any polynomial function poly(·), any memory location L ∈ [M], and any pair of messages (m₀, m₁) ∈ {0, 1}^κ × {0, 1}^κ that

\[ \Pr \left[ \begin{array}{c} m = m_D[L] \\
\text{crs} \leftarrow \text{crsGen}(1^\kappa) \\
\text{digest}, D \leftarrow \text{hash}(\text{crs}, D) \\
\text{ct} \leftarrow \text{Send}(\text{crs}, \text{digest}, L, m₀, m₁) \\
m \leftarrow \text{Receive}(\text{crs}, \text{ct}, L) \end{array} \right] = 1, \]

where the probability is taken over the random choices made by crsGen and Send.
– **Sender Privacy Against Semi-Honest Receivers:** There exists a PPT simulator $\ell OT.\text{Sim}$, such that for all PPT adversaries $A$, the following holds

$$\text{REAL}_A(1^\kappa) \approx \text{IDEAL}_A^{\ell OT.\text{Sim}}(1^\kappa)$$

where the experiments are defined as follows:

**Experiment** $\text{REAL}_A(1^\kappa)$:
- $\text{crs} \leftarrow \text{crsGen}(1^\kappa)$
- $(\text{aux}, m_0, m_1, D, L) \leftarrow A(\text{crs})$
- $\text{digest} := \text{hash}(\text{crs}, D)$
- $\text{ct} \leftarrow \text{Send}(\text{crs}, \text{digest}, L, m_0, m_1)$
- Output $(\text{aux}, \text{ct})$

**Experiment** $\text{IDEAL}_A^{\ell OT.\text{Sim}}(1^\kappa)$:
- $\text{crs} \leftarrow \text{crsGen}(1^\kappa)$
- $(\text{aux}, m_0, m_1, D, L) \leftarrow A(\text{crs})$
- $\text{ct}' \leftarrow \ell OT.\text{Sim}(\text{crs}, D, L, m_D[L])$
- Output $(\text{aux}, \text{ct}')$

In the above, database $D$ should be of size at most $M = \text{poly}(\kappa)$, for any polynomial function $\text{poly}(\cdot)$, memory location $L \in [M]$, and messages $(m_0, m_1) \in \{0, 1\}^\kappa \times \{0, 1\}^\kappa$.

– **Efficiency Requirement:** The length of digest is a fixed polynomial in $\kappa$ independent of the size of the database; we will assume for simplicity that $|\text{digest}| = \kappa$. Moreover, the algorithm hash runs in time $|D|.\text{poly}(\log |D|, \kappa)$, Send and Receive run in time $\text{poly}(\log |D|, \kappa)$.

\[\triangleleft\]

**Definition 15 (One-Time Signature).** A one-time signature (OTS) scheme $\text{OTS}$ for message space $M = \{0, 1\}^n$ consists of three algorithms as follows:

– $\text{OTS}.\text{gen}(1^\kappa) \rightarrow (\text{OTS}.\text{vk}, \text{OTS}.\text{sk})$: On input security parameter $\kappa$, outputs a pair $(\text{OTS}.\text{vk}, \text{OTS}.\text{sk})$ of verification and signing keys.

– $\text{OTS}.\text{sign}(\text{OTS}.\text{sk}, x) \rightarrow \tau$: On input a signing key $\text{OTS}.\text{sk}$ and message $x$, outputs a signature $\tau$.

– $\text{OTS}.\text{verify}(\text{OTS}.\text{vk}, \tau, x) \rightarrow b'$: On input verification key $\text{OTS}.\text{vk}$, signature $\tau$ and a message $x$, outputs a bit $b'$.

where they satisfy the following properties.

1. **Succinctness:** For $(\text{OTS}.\text{vk}, \text{OTS}.\text{sk}) \leftarrow \text{OTS}.\text{gen}(1^\kappa)$ it holds that $|\text{OTS}.\text{vk}|$ is independent of $n$ and only depends on $\kappa$.

2. **Correctness of Verification:** $\forall \kappa, x \in M$, it holds that if $(\text{OTS}.\text{vk}, \text{OTS}.\text{sk}) \leftarrow \text{OTS}.\text{gen}(1^\kappa)$, then

$$\text{OTS}.\text{verify}(\text{OTS}.\text{vk}, \text{OTS}.\text{sign}(\text{OTS}.\text{sk}, x), x) = 1$$

3. **Unforgeability:** For any PPT adversary $A$, there exists a negligible function $\text{negl}(\cdot)$ such that the following holds:

$$\Pr[\text{SIG-forgel-TIME}(A) = 1] \leq \text{negl}(\kappa)$$

where $\text{SIG-forgel-TIME}$ is shown in Figure 10.

\[\triangleleft\]
Definition 16 (One-Time Signature with Encryption). A one-time signature with encryption (OTSE) scheme is a one-time signature (OTS) scheme that consists of additional algorithms as follows:

- **OTSE.\text{enc}(OTSE.vk, i, b, m) \rightarrow ct:** On input verification key \(OTSE.vk\), a bit position \(i\), a bit \(b\) and plaintext \(m\), outputs a ciphertext \(ct\).
- **OTSE.\text{dec}(OTSE.vk, \tau, x, ct) \rightarrow m:** On input verification key \(OTSE.vk\), message \(x\), signature \(\tau\) and ciphertext \(ct\), outputs a plaintext \(m\).

Where they satisfy the following properties.

1. **Correctness of encryption:** For all \(\kappa, x, m\), it holds that if \((OTSE.vk, OTSE.sk) \leftarrow OTSE.\text{gen}(1^\kappa)\) and \(\tau \leftarrow OTSE.\text{sign}(OTSE.sk, x)\); then

\[
\text{OTSE.\text{dec}(OTSE.vk, \tau, x, OTSE.\text{enc}(OTSE.vk, i, x_i, m))} = m
\]

2. **Selective security of encryption:** For any PPT adversary \(A = (A_1, A_2, A_3)\), there exists a negligible function \(\text{negl}(\cdot)\) such that the following holds:

\[
\Pr[\text{IND}^\text{OTSE}(A) = 1] \leq \frac{1}{2} + \text{negl}(\kappa)
\]

where \(\text{IND}^\text{OTSE}\) is shown in Figure 11.

\(\triangleright\)

Definition 17 (Signature scheme). A signature scheme \(\text{Sig}\) for message space \(M = \{0, 1\}^n\) consists of three algorithms as follows:

- **\text{Sig.\text{gen}(1^\kappa) \rightarrow (Sig.vk, Sig.sk):** On input security parameter \(\kappa\), outputs a pair \((Sig.vk, Sig.sk)\) of verification and signing keys.
- **\text{Sig.\text{sign}(Sig.sk, x) \rightarrow \tau:}\** On input a signing key \(Sig.sk\) and message \(x\), outputs a signature \(\tau\).
- **\text{Sig.\text{verify}(Sig.vk, \tau, x) \rightarrow b':}\** On input verification key \(Sig.sk\), signature \(\tau\) and a message \(x\), outputs a bit \(b'\).

where they satisfy the following properties.

1. **Succinctness:** For \((Sig.vk, Sig.sk) \leftarrow \text{Sig.\text{gen}(1^\kappa)}\) it holds that \(|\text{Sig.vk}|\) is independent of \(n\) and only depends on \(\kappa\).
Experiment IND^{OTSE}

Parameter: Let $\kappa$ be the security parameter.
- $(s_{A_1}, x) \leftarrow A_1(1^\kappa)$.
- $(\text{OTSE} . vk, \text{OTSE} . sk) \leftarrow \text{OTSE} . \text{gen}(1^\kappa)$.
- $\tau \leftarrow \text{OTSE} . \text{sign}(\text{OTSE} . sk, x)$.
- $(s_{A_2}, i, m_0, m_1) \leftarrow A_2(s_{A_1}, \text{OTSE} . vk, \tau)$.
- $b^* \leftarrow \{0, 1\}$.
- $m^* := m_{b^*}$.
- $c^* \leftarrow \text{OTSE} . \text{enc}(\text{OTSE} . vk, i, 1 - x_i, m^*)$.
- $b' \leftarrow A_3(s_{A_2}, c^*)$.
- Output 1 if $b' = b^*$ and 0 otherwise.

Fig. 11. One-time Signature with Encryption ind-security Experiment.

2. Correctness of verification: \(\forall \kappa, x \in M\), it holds that if \((\text{Sig} . vk, \text{Sig} . sk) \leftarrow \text{Sig} . \text{gen}(1^\kappa)\); then
\[
\text{Sig} . \text{verify}(\text{Sig} . vk, \text{Sig} . \text{sign}(\text{Sig} . sk, x), x) = 1
\]

3. Unforgeability: For any PPT adversary $A$, there exists a negligible function $\text{negl}(.)$ such that the following holds:
\[
\Pr[\text{Sig}-\text{forge}(A) = 1] \leq \frac{1}{2} + \text{negl}(\kappa)
\]

where $\text{Sig}-\text{forge}$ is shown in Figure 12.

Experiment SIG-forge

Parameter: Let $\kappa$ be the security parameter.
- $(\text{Sig} . vk, \text{Sig} . sk) \leftarrow \text{Sig} . \text{gen}(1^\kappa)$.
- $(m, \tau) \leftarrow A^{\text{Sig} . \text{sign}(\text{Sig} . sk,.)(\text{Sig} . vk)}$.
- If $A$ did not query the oracle $\text{Sig} . \text{sign}(\text{Sig} . sk,.)$ for signature of $m$ and $\text{Sig} . \text{verify}(\text{Sig} . vk, \tau, m) = 1$, output 1.
  Else, output 0.

Fig. 12. Signature unforgeability security Experiment.

Definition 18 (Puncturable Signature). A puncturable signature scheme $\text{Sig}$ is a signature scheme that consists of the following additional algorithms:
- \text{Sig.punct}(\text{Sig.sk}, \text{mpre}) \rightarrow \text{Sig.sk}': On input a signing key \text{Sig.sk} and a message prefix \text{mpre}, outputs a punctured signing key \text{Sig.sk}'.
- \text{Sig.psign}(\text{Sig.sk}', m) \rightarrow \tau: On input a punctured signing key \text{Sig.sk}' and a message \text{m}, outputs a signature \tau.

where they satisfy the following properties.

1. \textbf{Correctness of verification}: \( \forall \kappa, \text{prefix mpre and } x \in M, \text{it holds that if } (\text{Sig.vk}, \text{Sig.sk}) \leftarrow \text{Sig.gen}(1^\kappa), \text{Sig.sk}' \leftarrow \text{Sig.punct}(\text{Sig.sk}, \text{mpre}), \text{and } x \text{ does not have mpre as prefix, then} \\
\text{Sig.verify}(\text{Sig.vk}, \text{Sig.psign}(\text{Sig.sk}', x), x) = 1 \)

2. \textbf{Unforgeability}: For any PPT adversary \( \mathcal{A} = \{ \mathcal{A}_1, \mathcal{A}_2 \} \), there exists a negligible function \( \text{negl}(\cdot) \) such that the following holds:
\[
\Pr[\text{SIG-forge}_\text{PUNCT}(\mathcal{A}) = 1] \leq \frac{1}{2} + \text{negl}(\kappa)
\]

where \( \text{SIG-forge}_\text{PUNCT} \) is shown in Figure 13.

\begin{center}
\textbf{Experiment SIG-forge}_\text{PUNCT}
\end{center}

\textbf{Parameter:} Let \( \kappa \) be the security parameter.

- \((\text{Sig.vk}, \text{Sig.sk}) \leftarrow \text{Sig.gen}(1^\kappa)\).
- \((\text{mpre}, s_{\mathcal{A}_1}) \leftarrow A_1^\text{Sig.sign}(\text{Sig.sk}.)(\text{Sig.vk})\).
- \text{Sig.sk}' \leftarrow \text{Sig.punct}(\text{Sig.sk}, \text{mpre}).
- \((m, \tau) \leftarrow A_2^\text{Sig.sign}(\text{Sig.sk}.)(s_{\mathcal{A}_1}, \text{Sig.sk}').\)
- If \text{mpre} is a message prefix of \text{m}, and \( \mathcal{A}_1, \mathcal{A}_2 \) queried the oracle \text{Sig.sign}(\text{Sig.sk}.,) for signature of at most one message \text{m}' s.t. \text{mpre} is a message prefix of \text{m}' and \( m \neq m' \), and \text{Sig.verify}(\text{Sig.vk}, \tau, m) = 1 \), output 1.
- Else, output 0.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Fig_13.png}
\caption{Puncturable Signature unforgeability security Experiment.}
\end{figure}

\begin{definition}[(Prefix) Puncturable Pseudorandom Function Family]. A prefix puncturable pseudorandom function family scheme (PPRF) consists of the following PPT algorithms.

- \( F.\text{gen}(1^\kappa) \rightarrow s: \) On input the security parameter, outputs a key \( s \).
- \( F.\text{punct}(s, \text{pre}) \rightarrow s^*: \) On input a key \( s \) and a prefix \( \text{pre} \), outputs a punctured key \( s^* \).
- \( F.\text{eval}(s, x) \rightarrow y: \) On input a key \( s \) and a string \( x \), outputs a string \( y \).

The following properties are required. \textbf{Pseudorandomness}: For all PPT adversaries \( \text{Adv} \), it holds that:
\[
\Pr[\text{Adv}^F.\text{eval}(s,. \text{ outputs 1})] - \Pr[\text{Adv}^R(\cdot \text{ outputs 1})] \leq \text{negl}(\kappa)
\]
\end{definition}
where, $s \leftarrow F.\text{gen}(1^n)$ and $R(.)$ is a random function. **Correctness of Puncturing:** \( \forall s, \text{pre} \), it holds that

$$F.\text{eval}(F.\text{punct}(s, \text{pre}), x) = \begin{cases} F.\text{eval}(s, x) & \text{if } x \text{ does not have prefix } \text{pre} \\ \bot & \text{else} \end{cases}$$

**Security of Puncturing:** For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\text{negl}(\cdot)$ such that the following holds

$$\Pr[\text{IND}^{\text{PPRF}}(\mathcal{A}) = 1] \leq \frac{1}{2} + \text{negl}(\kappa)$$

where $\text{IND}^{\text{PPRF}}$ is shown in Figure 14.

---

**Experiment $\text{IND}^{\text{PPRF}}$**

**Parameter:** Let $\kappa$ be the security parameter.

- $s \leftarrow F.\text{gen}(1^n)$.
- $(\text{st}_{\mathcal{A}_0}, \text{pre}) \leftarrow \mathcal{A}_0.\text{eval}(s, \cdot)(1^n)$.
- $s^* \leftarrow F.\text{punct}(s, \text{pre})$.
- $(\text{st}_{\mathcal{A}_1}, x) \leftarrow \mathcal{A}_1.\text{eval}(s^*, \cdot)(\text{st}_{\mathcal{A}_0}, s^*)$.
- $b \leftarrow \{0, 1\}$.
  - If $b = 0$, set $y \leftarrow F.\text{eval}(s, x)$, else uniformly sample $y \leftarrow \mathcal{Y}$.
  - $b^* \leftarrow \mathcal{A}_2.\text{eval}(s^*, \cdot)(\text{st}_{\mathcal{A}_1}, y)$.
- Output 1 if $b^* = b$ and $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2$ did not query oracle $F.\text{eval}(\cdot, \cdot)$ for any $x'$ with prefix $\text{pre}$, else 0.

---

**Fig. 14.** Puncturable PRF Security Experiment.

---

**B Proof of Theorem 1**

**Theorem 1 (Restated).** Suppose $\mathcal{G}$ is a $(\mathcal{P}, \mathcal{M})$-generator class that is decomposable into $\mathcal{L} = \{\mathcal{H}_i, Q_i, \mathcal{P}_i\}_{i \in [N]}$, such that, for each $i \in [N]$, $\mathcal{H}_i$ is a $(\mathcal{P}_i, \mathcal{M}_i)$-generator class and there exists an R3PO scheme $\mathcal{O}_i$ for $(\mathcal{H}_i, Q_i, \mathcal{P}_i)$. Then there exists an R3PO scheme $\mathcal{O}$ for $(\mathcal{G}, Q, \mathcal{P})$ where $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_N$.

*Proof Sketch.* The obfuscator $\mathcal{O}$ is shown in Figure 3. We prove that this is a valid obfuscation scheme for every $(\mathcal{P}, \mathcal{M})$-generator class $\mathcal{G}$ via a sequence of hybrids. Fix any $\mathcal{G} \in \mathcal{G}$ and any $Q \in \mathcal{Q}$. For each $i \in \{1, 2, \cdots, N\}$, we define a sequence of hybrid experiments $\text{Hybrid}^{1,0}$ through $\text{Hybrid}^{i,0}$, and argue indistinguishability of the outputs from adjacent hybrids, and further show that the outcome of $\text{Hybrid}^{i,0}$ is indistinguishable from that of $\text{Hybrid}^{i+1,0}$. Finally, the outcome of the real obfuscation experiment will be shown to be identical to that of $\text{Hybrid}^{1,0}$ and that of the ideal obfuscation experiment will be shown to be identical to that of $\text{Hybrid}^{N+1,0}$.

For each $i \in [N]$, as we go from $\text{Hybrid}^{i,0}$ to $\text{Hybrid}^{i+1,0}$, we build a successively increasing $(i+1)$-partial reach-extractor $E_i$ and a corresponding partial simulation. $E_i$ is built recursively from an $i$-partial reach-extractor $E_{i-1}$ and a one-step extractor $E_i$ corresponding to $\mathcal{H}_i$ w.r.t. $Q$. This is immediate from the
tree-ordering property of reactive programs in $\mathcal{P}$\(^{17}\). The corresponding partial simulation will simulate the garbled circuit for partition $i$ on a simulated obfuscation of a one-step $\sigma_i$-restricted program, where $\{\sigma_i\}$ is as extracted by $E_{i-1}$ and bounds the reach of the adversary in partition $i$.

- **Hybrid\(^{i,0}\):** In this hybrid, the reactive program is generated by $G$ after interacting with the adversary $Q\|E_{i-1}$. All garbled circuits up to partition $i-1$ and the one-step obfuscated programs output by them are simulated. The garbled circuits for partitions $i, \ldots, N$ are as in the real obfuscation $O$.

- **Hybrid\(^{i,1}\):** In this hybrid, the $i$th garbled circuit is simulated on an input $\text{stt}_i$ extracted by $E_{i-1}$, but with a real output $f_i(\sigma_i)$ (that will have labels for all states reachable from $\sigma_i$). Indistinguishability follows from a combination of reach restriction of $E_{i-1}$ and garbled circuit security.

- **Hybrid\(^{i,2}\):** In this hybrid, we augment the interaction with additional programs corresponding to the decomposition at partition $i$. That is, we let $G\|J_i$ interact with $Q\|E_{i-1}[W_i]$. $J_i$ outputs a one-step $\sigma_i$-restricted reactive program, where $\sigma_i$ is the state in partition $i$ reachable by $E_{i-1}$. Indistinguishability holds since the interaction between $G$ and $Q\|E_{i-1}$ is unchanged, and so is the rest of the experiment.

- **Hybrid\(^{i,3}\):** In this hybrid, we use the decomposition of $G$ to $L_i$ to have $H_i\|Z_i$ interact with $Q\|E_{i-1}[W_i]$. This finishes the simulation of the $i$th garbled circuit. Indistinguishability follows from R3PO security of $O$.

- **Hybrid\(^{i,4}\):** In this hybrid, we use the one-step extractor $E_i$ and obfuscation simulator $\hat{\text{Sim}}_i$ for $H_i$ from the library to simulate the $f_i(\sigma_i)$ (obfuscation of the one-step program at $\sigma_i$). This finishes the simulation of the $i$th garbled circuit. Indistinguishability holds even if a passive program $E_i$ is allowed to see inside $W_i$.

- **Hybrid\(^{i,5}\):** In this hybrid, we move a part of $J_i$ to the other side to get back an interaction between $G$ and adversary $Q$ composed with a new partial reach-extractor $E_i$ (defined in terms of $E_{i-1}[W_i]|E_i$ and $J_i$). Indistinguishability with Hybrid\(^{i,5}\) follows from the properties of $J_i$. Also hybrid Hybrid\(^{i,6}\) and Hybrid\(^{i+1,0}\) are identical to each other.

**Proof:** We first state some key observations that will be useful in interpreting the hybrids:

- Programs $J_i$ and $K_i$. Let the simulators corresponding to the decomposition (Definition 5) of $G, Q$ to $L_i$ at any partition $i$ be $J_i, W_i, Z_i$. We redefine $J_i$ as a composition of two programs $J_i$ and $K_i$, s.t. $K_i$ is a non-interacting program that uniformly samples $\tilde{\mu}_i^{(\beta)}$ and outputs it at the end of the interaction. This corresponds to the fact that $J_i$ samples $\tilde{\mu}_i^{(\beta)}$ uniformly at the end of the interaction.

- Partial reach-extractor $E_i$. For any $i$, as we go from hybrid Hybrid\(^{i,0}\) to Hybrid\(^{i+1,0}\), we will be building an $i$-partial reach-extractor recursively from an $i-1$-partial reach-extractor $E_{i-1}$ and an one-step extractor $E_i$ as follows. $E_{i-1}$ outputs $(\Pi_{i-1}, X^{*,i-1})$ and $E_i$ outputs $(\hat{\Pi}_i, \hat{X}_i^*)$; then, output of $E_i$ is defined as $\Pi^i = \Pi^{i-1} \times \hat{\Pi}_i, X^{*,i} = X^{*,i-1} \times \hat{X}_i^*$. The partial extractor $E_0$ simply outputs $a_Q$.

- **Sampling the labeling function $\hat{\beta}$:**
  - Hybrids with generator $G$ sample the labeling function $\hat{\beta}$ uniformly at random outside the interaction.
  - Hybrids with generator $H_i\|Z_i$ has $K_i$ sample $\tilde{\beta}(k, \ldots)$ for $k \in [i, N]$ uniformly at random. The labeling functions $\tilde{\beta}(k, \ldots)$ for $k < i$ are sampled uniformly at random outside the interaction.
  - Hybrids with generator $H_i\|Z_i$ has $K_i$ sample $\tilde{\beta}(k, \ldots)$ for $k \in [i, N]$ uniformly at random. The labeling functions $\tilde{\beta}(k, \ldots)$ for $k < i$ are sampled uniformly at random outside the interaction.

In all the three cases above, the distribution of $\hat{\beta}$ does not change.

\(^{17}\) there is at most a single partition $k < i$ from which states in partition $i$ can be transitioned to. Then, the extraction corresponds to using the one-step extractor $E_k$.
Reach-extractor and obfuscation simulator for $G$. As we prove the theorem, we also construct the extractor and simulator that satisfy the R3PO definition. Looking ahead, the reach-extractor for $G$ w.r.t. the adversary $Q$ corresponds to $E_{N+1}$. Similarly, the R3PO simulator is essentially $\text{Hybrid}^{N+1,0}$, excluding the interaction.

- **Hybrid$^{i,0}$:** In this hybrid, the reactive program is generated by $G$ while interacting with the adversary $Q^\top | E_{i-1}$, where $E_{i-1}$ is empty for $i = 1$ and the composite machine $(E_{i-2} | W_{i-1} | E_i || J_{i-1})$ for $i > 1$. We show in Lemma 6 that $E_{i-1}$ is a $i$-partial reach-extractor for $Q$ w.r.t. $G$.

  \[
  \begin{align*}
  \text{Hybrid}^{i,0}: \\
  &- (\pi^{(a)}, \mu^{(b)}, a_G; a_Q, \Pi^{i-1}, X^{*,i-1}) \leftarrow \langle G : Q | E_{i-1} \rangle \\
  &- \text{Sample } \beta. \\
  &- \text{Define } \Sigma_{rch} = \text{REACH}_{\Pi_{i-1}}(X^{*,i-1}) \text{ and extract } \{\overline{\Pi}_j, X^*\}_j \text{ from } (\Pi^{i-1}, X^{*,i-1}). \\
  &- \text{For each } j \in [N], \text{ define functions } \overline{\mu}_j(\beta), f_j \text{ as follows:} \\
  &- \forall \sigma' \in \Sigma, \quad \overline{\mu}_j(\beta)(\sigma') := \text{encode}_{\beta,j}^x(\sigma') \\
  &- \forall \sigma \in \Sigma, \quad f_j(\sigma) := \left(\overline{G}_j(\overline{\pi}_x^{(a)}, \overline{\mu}_j(\beta)), \mu(\beta)(\sigma)\right) \\
  &- \text{For each } j \in [N] \text{ such that } j < i: \\
  &- \sigma_j = \Sigma_{rch} \cap \Sigma_j \\
  &- \text{If } \sigma_j = \bot, \text{ then } \overline{GC}_j \leftarrow \text{GCircSim}(\varphi(f_j)) \\
  &- \text{Else:} \\
  &- \quad \text{Sample } \overline{\rho}_j \leftarrow \text{Sim}_j(\overline{\pi}_x^{(a)}, \overline{\mu}_j, \overline{\Pi}_j, \{x, \overline{\rho}_j(\text{REACH}_{\overline{\Pi}_j}(x))\}_{x \in X^*_j}) \\
  &- \quad \text{Let } y_{\sigma_j} = (\overline{\rho}_j, \mu(\beta)(\sigma_j)), \text{lab}_{\sigma_j} = \text{encode}_{\beta,j}^{x,i-1}(\sigma_j) \\
  &- \quad \text{If } j > 1, \text{ sample } \overline{GC}_j \leftarrow \text{GCircSim}(\varphi(f_j), \text{lab}_{\sigma_j}, y_{\sigma_j}) \\
  &- \quad \text{For each } 1 \leq j \leq N \text{ such that } j \geq i, \text{ sample } \overline{GC}_j \leftarrow \text{GCircSim}(f_j, \overline{\beta}_j) \\
  &- \quad \text{If } i \in \{1, 2\}, \text{ set } \{y_{\sigma_1}, \{\overline{GC}_j\}_{1 \leq j \leq N}\} \\
  &- \quad \text{else } \rho = \left(y_{\sigma_1}, \{\overline{GC}_j\}_{1 < j < i}, \{\overline{GC}_j\}_{N \geq j \geq i}\right) \\
  &- \quad \text{Output } (\alpha_G, a_Q, \rho)
  \end{align*}
\]

- **Hybrid$^{i,1}$:** In this hybrid, the $i^{th}$ garbled circuit is simulated given an input $\sigma_i$, input labels $\text{encode}_{\beta,j}^{x,i}(\sigma_i)$ and output $f_i(\sigma_i)$; where, $\sigma_i = \Sigma_{rch} \cap \Sigma_j$ is the state extracted by the $i$-partial reach-extractor $E_{i-1}$. **Indistinguishability with Hybrid$^{i,0}$:** Note that, from the reach bound and reach restriction of $E_{i-1}$, an adversary can evaluate the garbled circuit $G_{C_i}$ in Hybrid$^{i,0}$ at most a single point (corresponding to the state reached: $\sigma_i$). Thus, from a reduction to garbled circuit security, the indistinguishability follows.
Hybrid\textsuperscript{i,1}:
- \((\pi^{(\alpha)}, \mu^{(\beta)}, a_G; a_Q, \Pi^{i-1}, X^{*,i-1}) \leftarrow \langle G : Q \rangle_{E_{i-1}}\)
- Sample \(\tilde{\beta}\).
- Define \(\Sigma_{rch} = \text{REACH}_{\Pi^{i-1}}(X^{*,i-1})\) and extract \(\{\pi_j, \tilde{X}_j\}\)\(_{1 \leq j < i}\) from \((\Pi^{i-1}, X^{*,i-1})\).
- For each \(j \in [N]\), define functions \(\mu_j(\beta), f_j\) as follows:
  - \(\forall \sigma' \in \Sigma, \mu_j(\beta)(\sigma') := \text{encode}_{j-1}\(\beta\)(\sigma')\)
  - \(\forall \sigma \in \Sigma, f_j(\sigma) := (O_j(\pi^{(\alpha)}_j, \mu_j(\beta)), \mu(\beta)(\sigma))\)
- For each \(j \in [N]\) such that \(j \leq i\):
  - \(\sigma_j = \Sigma_{rch} \cap \Sigma_j\)
  - If \(\sigma_j = \bot\), then \(\overline{G}C_j \leftarrow \text{G CircSim}(\Phi(f_j))\)
  - Else:
    - If \(j < i\), sample \(\tilde{\rho}_j \leftarrow \text{Sim}_j(\pi^{(\alpha)}_j, \mu^{(\beta)}_j, \pi_j, \{x, \mu^{(\beta)}_j(\text{REACH}_j(x))\}_{x \in \tilde{X}_j})\)
        and set \(y_{\sigma_j} = (\tilde{\rho}_j, \mu^{(\beta)}(\sigma_j))\).
    - If \(j = i\), set \(\rho_j = O_j(\pi^{(\alpha)}_j, \mu_j(\beta))\) and \(y_{\sigma_j} = (\rho_j, \mu^{(\beta)}(\sigma_j))\)
    - If \(j \leq i\), \(\text{lab}_{\sigma_j} = \text{encode}_{j-1}\(\beta\)(\sigma_j)\)
    - If \(j > 1\), sample \(\overline{G}C_j \leftarrow \text{G CircSim}(\Phi(f_j), \text{lab}_{\sigma_j}, \rho_{\sigma_j})\)
- For each \(j \in [N]\) such that \(j > i\), sample \(GC_j \leftarrow \text{GC Garble}(f_j, \tilde{\beta}_j)\)
- If \(i \in \{1, 2\}\), set \((y_{\sigma_i}, \{GC_j\})_{1 < j \leq N}\)
  - Else \(\rho = (y_{\sigma_i}, \{GC_j\})_{1 < j \leq N}, \{GC_j\}_{N \geq j \geq i})\)
  - Output \((a_G, a_Q, \rho)\)

- Hybrid\textsuperscript{i,2}: In this hybrid, we let \(G \| \mathcal{J}_i \| \mathcal{K}_i\) interact with \(\langle Q \rangle_{E_{i-1}}|W_i\), where \(\mathcal{J}_i, \mathcal{K}_i, W_i\) are as specified by the decomposition of \(G\) to \(\mathcal{L}_i\) at partition \(i\). Indistinguishability with Hybrid\textsuperscript{i,1}: \(\mathcal{J}_i, \mathcal{K}_i\) and \(W_i\) are such that, they are passive to the interaction between \(G\) and \(Q\). \(\mathcal{K}_i\) uniformly samples \(\tilde{\beta}(k, \ldots)\) for \(k \in [i, N]\), but only outputs it at the end of the interaction. Thus, we have that the distributions of \((\pi^{(\alpha)}, \mu^{(\beta)}, a_G, \tilde{\beta})\) and output of \(E_{i-1}\) are identical in both the hybrids. Hence the hybrids are identical. From the correctness of decomposition (Definition 5), \(\mathcal{J}_i\) outputs a one-step restriction at state \(\sigma_i\) consistent with the extraction of \(E_{i-1}\) (which is a \(i\)-partial reach-extractor).
Hybrid$^i$·2:
- \((\bar{\pi}_i^{(\alpha)}, \bar{\mu}_i^{(\beta)}), \pi_i^{(\alpha)}, \mu_i^{(\beta)}, a_G, a_Q, \Pi_{i-1}^i, X_{i}^{*,i-1}) \leftarrow (G)_{J_i} (K_i : Q | E_{i-1} | W_i) \)
- Sample \(\bar{\beta}\) consistent with \(\bar{\mu}_i^{(\beta)}\) (that is, sample \(\bar{\beta}(k_{i-1})\) for \(k < i\).
- Define \(\Sigma_{rch} = \text{REACH}_{i-1}(X_{i}^{*,i-1})\) and extract \(\{\bar{\Pi}_j, \bar{X}_j\}_{1 \leq j < i}\) from \((\Pi_{i-1}^i, X_{i}^{*,i-1})\).
- For each \(j \in [N]\), define functions \(\bar{\mu}_j^{(\beta)}, f_j\) as follows:
  - \(\forall \sigma' \in \Sigma, \quad \bar{\mu}_j^{(\beta)}(\sigma') := \text{encode}_{\beta}^j(\sigma')\)
  - \(\forall \sigma \in \Sigma, \quad f_j(\sigma) := \left(\sigma_j(\bar{\pi}_j^{(\alpha)}, \bar{\mu}_j^{(\beta)}), \mu_j^{(\beta)}(\sigma)\right)\)
- For each \(j \in [N]\) such that \(j \leq i\):
  - \(\sigma_j = \Sigma_{rch} \cap \Sigma_j\)
  - If \(\sigma_j = \bot\), then \(\text{GC}_j \leftarrow \text{GCircSim}(\Phi(f_j))\)
  - Else:
    - If \(j < i\), sample \(\bar{\rho}_j \leftarrow \text{Sim}_j(\bar{\pi}_j^{(\alpha)}, \bar{\mu}_j^{(\beta)}), \bar{\Pi}_j, \{x, \bar{\mu}_j^{(\beta)}(\text{REACH}_{\bar{\Pi}_j}(x))\}_{x \in \bar{X}_j}\)
      and set \(y_{\sigma_j} = (\bar{\rho}_j, \mu_j^{(\beta)}(\sigma_j))\).
    - If \(j = i\), set \(\bar{\rho}_j = \sigma_j(\bar{\pi}_j^{(\alpha)}, \bar{\mu}_j^{(\beta)})\) and \(\rho_j = (\mu_j^{(\beta)}(\sigma_j))\)
    - If \(j \leq i\), \(\text{lab}_{\sigma_j} = \text{encode}_{\beta}^{j-1}(\sigma_j)\)
    - If \(j > 1\), sample \(\text{GC}_j \leftarrow \text{GCSim}(\Phi(f_j), \text{lab}_{\sigma_j}, y_{\sigma_j})\)
- For each \(j \in [N]\) such that \(j > i\), sample \(\text{GC}_j \leftarrow \text{GCGarble}(f_j, \bar{\beta}_j)\)
  - If \(i \in \{1, 2\}\), set \(\left(y_{\sigma_1}, \{\text{GC}_j\}_{1 \leq j \leq N}\right)\)
  - Else \(\rho = \left(y_{\sigma_1}, \{\text{GC}_j\}_{1 \leq j < i}, \{\text{GC}_j\}_{N \geq j \geq i}\right)\)
  - Output \((a_G, a_Q, \rho)\)

Hybrid$^i$·3: In this hybrid, we use the interaction \(\langle H_i \| Z_i : Q | E_{i-1} | W_i \rangle\) to generate the reactive program, where \(H_i\) corresponds to the decomposition of \(G\) to \(L_i\). Indistinguishability with Hybrid$^i$·2 trivially follows: corresponding to \(Q, E_{i-1}, J_i, K_i, W_i\) of the previous hybrid, \(Z_i\) is such that the distributions \(\langle G \| J_i \| K_i : Q | E_{i-1} | W_i \rangle\) and \(\langle H_i \| Z_i : Q | E_{i-1} | W_i \rangle\) are indistinguishable.
Hybrid\textsuperscript{i\textasciicircum{3}}:

- \((\hat{\pi}_i, \hat{\mu}_i, a_Q, \Pi^{i-1}, X^{*,i-1}) \leftarrow H_i \parallel Z_i : Q|E_{i-1}|W_i\)
- Parse \(a_Q\) as \((\hat{\pi}(a), \mu(a), a_G)\).
- Sample \(\hat{\beta}\) consistent with \(\hat{\mu}_i\) (that is, sample \(\hat{\beta}(\hat{a})\) for \(k < i\)).
- Define \(\Sigma_{reach} = \text{REACH}_{i-1}(X^{*,i-1})\) and extract \(\{\hat{\Pi}_j, \hat{X}_j\}_{1 \leq j \leq i}\) from \((\Pi^{i-1}, X^{*,i-1})\).
- For each \(j \in [N]\), define functions \(\hat{\mu}_j, f_j\) as follows:
  - \(\forall \sigma' \in \Sigma, \: \hat{\mu}_j(\sigma') := \text{encode}_{\hat{\beta}}(\sigma')\)
  - \(\forall \sigma \in \Sigma, \: f_j(\sigma) := \left(\Omega_j(\hat{\pi}_j), \hat{\mu}_j(\sigma)\right)\)
- For each \(j \in [N]\) such that \(j \leq i\):
  - \(\sigma_j = \Sigma_{reach} \cap \Sigma_j\)
  - If \(\sigma_j = \bot\), then \(G_{\text{CIRC}} \leftarrow \text{GCIRC}(\Phi(f_j))\)
  - Else:
    - If \(j < i\), sample \(\hat{\rho}_j \leftarrow \text{Sim}_j\left(\hat{\pi}_j, \Pi_j, \{x, \hat{\mu}_j(\text{REACH}_{\Pi_j}(x))\}_{x \in X_j}\right)\)
      and set \(\gamma_{\sigma_j} = (\hat{\rho}_j, \mu(\sigma_j))\).
    - If \(j = i\), set \(\rho_j = \Omega(\hat{\pi}_j, \hat{\mu}_j)\) and \(\gamma_{\sigma_j} = (\rho_j, \mu(\sigma_j))\).
    - If \(j \leq i\), \(lab_{\sigma_j} = \text{encode}_{\hat{\beta}}(\sigma_j)\)
    - If \(j > 1\), sample \(G_{\text{CIRC}} \leftarrow \text{GCIRC}((\Phi(f_j)), lab_{\sigma_j}, \gamma_{\sigma_j})\)
- For each \(j \in [N]\) such that \(j > i\), sample \(G_{\text{CIRC}} \leftarrow \text{GCIRC}(f_j, \hat{\beta}_j)\)
- If \(i \in \{1, 2\}\), set \(\gamma_{\sigma_1, \{G_{\text{CIRC}}\}_{1 \leq j \leq N}}\)
  - else \(\rho = \left(\gamma_{\sigma_1, \{G_{\text{CIRC}}\}_{1 \leq j \leq i}}, \{G_{\text{CIRC}}\}_{N \geq j \geq i}\right)\)
  - Output \((\hat{\pi}_G, a_Q, \rho)\)

- Hybrid\textsuperscript{i\textasciicircum{4}}: In this hybrid, we let \(H_i \parallel Z_i\) interact with \(Q|E_{i-1}|W_i|E_i\), where \(E_i\) is a reach-extractor for \(Q|E_{i-1}|W_{i-1}\) (such an extractor is guaranteed from the R3PO for \(L_i\)). Further, the \(i\textsuperscript{th}\) one-step obfuscation program is simulated using the R3PO simulator for \(H_i\) and \(O_{i}\). Indistinguishability with Hybrid\textsuperscript{i\textasciicircum{3}}: directly follows from from the one-step extractor guarantee and R3PO security of the one-step generator \(H_i \parallel Z_i\) (Definition 4).
Hybrid$^{4,5}$:
- $(	ilde{\pi}^{(a)}_i, \tilde{\mu}^{(3)}_i, a_{G_i}, a_Q, \Pi^{i-1}, X^{*,i-1}, (\tilde{\Pi}_i, \tilde{X}^{\ast}_i)) \leftarrow \langle H_i \| Z_i \| Q_i \| E_{i-1} \| W_i \| \xi_i \rangle$
- Parse $a_{G_i}$ as $(\pi^{(a)}_i, \mu^{(3)}_i, a_G)$.
- Sample $\tilde{\beta}$ consistent with $\tilde{\mu}^{(3)}_i$ (that is, sample $\tilde{\beta}(k, ..)$ for $k \neq i$).
- Define $\Sigma_{rch} = \text{REACH}_{\Pi^{i-1}}(X^{*,i-1}) \cup \text{REACH}_{\Pi^{i}}(\tilde{X}^{\ast}_i)$ and extract $\{(\tilde{\Pi}_j, \tilde{X}^{\ast}_j)\}_{1 \leq j \leq i}$ from $(\Pi^{i-1}, X^{*,i-1})$.
- For each $j \in [N]$, define functions $\tilde{\mu}^{(3)}_j, f_j$ as follows:
  - $\forall \sigma' \in \Sigma$, $\tilde{\mu}^{(3)}_j(\sigma') := \text{encode}^{\tilde{\beta}}_j(\sigma')$
  - $\forall \sigma \in \Sigma$, $f_j(\sigma) := (\tilde{\sigma}_j(\tilde{\pi}^{(a)}_j, \tilde{\mu}^{(3)}_j), \mu^{(3)}_j(\sigma))$
- For each $j \in [N]$ such that $j < i$:
  - $\sigma_j = \Sigma_{rch} \cap \Sigma_j$
  - If $\sigma_j = \bot$, then $\text{GC}_j \leftarrow \text{GC}_{\text{Sim}}(\Phi(f_j))$
- Else:
  - Sample $\tilde{\rho}_j \leftarrow \text{Sim}_j(\tilde{\pi}^{(a)}_j, \tilde{\mu}^{(3)}_j, \tilde{\Pi}_j, \{x, \tilde{\mu}^{(3)}_j(\text{REACH}_{\Pi^{i}}(x))\}_{x \in \tilde{X}^{\ast}_i})$
  - and set $y_{\sigma_j} = (\tilde{\rho}_j, \mu^{(3)}(\sigma_j)), \text{lab}_{\sigma_j} = \text{encode}^{\tilde{\beta}}_{\sigma_j-1}(\sigma_j)$
  - If $j > 1$, sample $\text{GC}_j \leftarrow \text{GC}_{\text{Sim}}(\Phi(f_j), \text{lab}_{\sigma_j}, y_{\sigma_j})$
  - For each $j \in [N]$ such that $j > i$, sample $\text{GC}_j \leftarrow \text{GC}_{\text{Garble}}(f_j, \tilde{\beta}_j)$
  - If $i \in \{1, 2\}$, set $\{\text{GC}_j\}_{1 \leq j \leq N}$
  - else $\rho = (y_{\sigma_1}, \{\text{GC}_j\}_{1 < j < i}, \{\text{GC}_j\}_{N \geq j \geq i})$
  - Output $(a_G, a_Q, \rho)$

- Hybrid$^{4,5}$: In this hybrid, we re-invoke the $j_i, k_i$ (used in the previous hybrids). **Indistinguishability with Hybrid$^{4,5}$**: We prove this via a reduction to the decomposition guarantee. Suppose the hybrids were distinguishable, then there exists a distinguisher (that simulates $E_i$ using the auxiliary output of $W_i$, which contains its entire state and view) and can break the decomposition.
Hybrid$^{i.6}$: We now move the sampling to outside the interaction (that is, remove $K_i$). It remains indistinguishable since $K_i$ was non-interacting and producing its output at the end of the interaction. Finally, we rewrite the interaction as $\langle G : Q|E_{i-1}|W_i|E_i||J_i \rangle$. *Indistinguishability with Hybrid$^{i.5}$*: this follows from the description of $J_i$, which we defined as only being allowed to see the interaction of $\langle G : Q \rangle$ (and not the output of $G$). *Indistinguishability with Hybrid$^{i+1.0}$*: we define $E_i = E_{i-1}|W_i|E_i||J_i$. Then, the hybrids are identical.
Let Com = (Setup, Commit, Open) be a UC-secure extractable commitment scheme (Definition 11). Let \( M \) be a message function class.
The Program family and Generator class. The generator class $\mathcal{G}_{\text{Com},M}^{\text{Com}}$ is a set of generators of the form $H\|Z$ for a fixed $H$ and all polynomial $Z$. The generator $H$ behaves as follows. It accepts a setup $\text{crs}$ and a commitment $c$ from $Q$. It also accepts a message function $\mu(\beta) \in \mathcal{M}$ from $Z$. It then outputs a reactive program $(\pi_{c,\text{crs}}, \mu(\beta))$, whose transition function w.r.t. a state space $\Sigma = \{\text{start}\} \cup \{\sigma_m | m \in M\}$ is specified as follows:

$$\pi_{c,\text{crs}}(\text{start}, d) = \begin{cases} \sigma_m & \text{if } d \text{ s.t. } \text{Open}(\text{crs}, c, d) = m \neq \bot \\ \text{start} & \text{otherwise}. \end{cases}$$

The partition function is defined as: $\mathcal{I}_{\text{Com}}(\text{start}) = 1$ and $\mathcal{I}_{\text{Com}}(\sigma_m) = 2$ for all $m \in M$.

Adversary class and Reach-Extractor. Let $\text{Setup}_{\text{Sim}}$ denote the setup algorithm used by a simulator for the security experiment of Com; then the adversary class is defined to have adversaries of the form $Q^T$, where $Q$ is an arbitrary PPT program, and $T \in \{\text{Setup}, \text{Setup}_{\text{Sim}}\}$. An adversary $Q^T$ runs $T$ honestly to get the $\text{crs}$, and sends it to the generator.

We define an extractor $\mathcal{E}$ as follows. It replaces the setup $T$ to $\text{Setup}_{\text{Sim}}$ and gets a trapdoor for the $\text{crs}$. It then runs the extractor of the commitment scheme to extract the message $m$ from the commitment $c$ output by $Q$. Note that, the extractor of the commitment scheme cannot directly extract the commitment opening $d$. Thus, $\mathcal{E}$ outputs $X^* = \{m\}$ and a relaxed transition function $\Pi_m \in \mathcal{P}_{\text{Com}}$ defined as: $\Pi_m(\text{start}, m') = \sigma_m$ if $m = m'$. It is easy to verify that $\mathcal{E}$ described above is a valid reach-extractor.

R3PO. We show that a commitment scheme $\text{Com}$ and an R3PO scheme w.r.t. $(\mathcal{G}_{\text{Com}}, \mathcal{Q}_{\text{Com}}, \mathcal{P}_{\text{Com}})$ can be built from any 2-round UC-secure OT scheme. If the OT scheme is semi-honest secure, then we instead get a weakly-secure commitment scheme. A weakly-secure commitment scheme $(\text{Commit}, \text{Open})$ has no setup, and the security requirements are a standard hiding property, and a weak binding property wherein the commitment phase is carried out honestly. Correspondingly, we get a R3PO for a commitment-opening generator class w.r.t. a semi-honest adversary class (that honestly commits to a valid string).

We now prove the following lemma.

**Lemma 1 (Restated).** Given a UC-secure 2-round OT scheme in the CRS model, there exists a UC-secure commitment scheme $\text{Com} = (\text{Setup}, \text{Commit}, \text{Open})$ and an R3PO scheme w.r.t. $(\mathcal{G}_{\text{Com}}, \mathcal{Q}_{\text{Com}}, \mathcal{P}_{\text{Com}})$, for any message function class $\mathcal{M}$.

Further, given a semi-honest secure 2-round OT scheme, there exists a weakly-secure commitment scheme $\text{Com} = (\text{Commit}, \text{Open})$ and an R3PO scheme w.r.t. $(\mathcal{G}_{\text{Com}}, \mathcal{Q}_{\text{Com}}, \mathcal{P}_{\text{Com}})$, for any message function class $\mathcal{M}$.

**Proof:** We prove this for the message function class $\hat{\mathcal{M}}$ (Definition 6). R3PO for any polynomial message function class can easily be derived by applying the composition theorem via a trivial decomposition to the same class but for $\hat{\mathcal{M}}$. We give the proof for the UC-secure version. In particular, we use a UC-secure 2-round OT scheme (Definition 12) to give a commitment scheme and a R3PO scheme w.r.t. $(\mathcal{G}_{\text{Com}}, \mathcal{Q}_{\text{Com}}, \mathcal{P}_{\text{Com}})$. It is easy to show that the scheme in Figure 15 is a valid commitment scheme (Definition 11). We now prove that obfuscator $O_{\text{Com}}$ in Figure 16 is a secure R3PO.

- **Correctness.** If input $d$ satisfies $\text{Com}.\text{Open}(\text{crs}, c, d) = x$, then from the correctness of the OT scheme, the evaluator can recover function values $\{\hat{\beta}(2, i, x_i)\}_{i \in \mathcal{I}}$ for $x$, which is equal to $\hat{\mu}(\hat{\beta})(\sigma_x)$.

- **Efficiency.** Follows from the efficiency of $(l \text{ instances of})$ OT scheme.

---

18 Formally, the weak binding property is that for all PPT $\mathcal{A}$, and $\forall m \in M$, $\Pr[\{c, d\} \leftarrow \text{Commit}(m; r), \text{Open}(c, \mathcal{A}(m, r)) = m', m' \notin \{m, \bot\}]$ is negligible, where the probability is over the choice of randomness $r$ used by Commit and the randomness of $\mathcal{A}$. The standard hiding property is that $\forall m_1, m_2 \in M$, Commit$(m_1)$ and Commit$(m_2)$ are computationally indistinguishable.
Commitment Scheme $\text{Com}$ from OT

**Security Parameter:** Let $\kappa$ be the security parameter.

**OT Scheme:** Let $\text{OT} = (\text{OT}.\text{Setup}, \text{OT}_1, \text{OT}_2, \text{OT}_3)$ be an OT scheme.

$\text{Com}$.Setup($1^\kappa$) $\rightarrow$ Com.cr$s$: Run $l$ instances of the OT.Setup($1^\kappa$) algorithm, i.e.,

$$\forall i \in [l], \text{OT}.cr$s$ $\leftarrow$ OT.Setup($1^\kappa$)

Set $\text{Com}.cr$s := \{ \text{OT}.cr$s$^i \}_{i \in [l]}$.

$\text{Com}.\text{Commit}(\text{Com}.cr$s$, $x$) $\rightarrow$ ($c$, $d$): Parse $\text{Com}.cr$s = \{ \text{OT}.cr$s$^i \}_{i \in [l]}$. Commit each bit of $x$ using the OT$_1$ algorithm:

$$\forall i \in [l], (\text{ots}_1^i, \omega') $\leftarrow$ OT$_1$(OT.cr$s$$_1^i$, $x_i$)

Set ($c$, $d$) := ($\{ \text{ots}_1^i \}_{i \in [l]}$, $\{ \omega' \}_{i \in [l]}$).

$\text{Com}.\text{Open}(\text{Com}.cr$s$, $c$, $d$) $\rightarrow$ $x$: Parse ($\text{Com}.cr$s$, $c$, $d$) as ($\{ \text{OT}.cr$s$^i \}_{i \in [l]}$, $\{ \text{ots}_1^i \}_{i \in [l]}$, $\{ \omega' \}_{i \in [l]}$). Open each bit of $x$ by using a OT$_3$ algorithm on a constructed OT$_2$ output as follows. For all $i \in [l]$, do

$$(m_0, m_1) \leftarrow \{0, 1\}^\kappa,$$

$$\text{ots}_2 \leftarrow \text{OT}_2(\text{OT}.cr$s$^i$, $\text{ots}_1^i$, $m_0$, $m_1$),$$

$$x_i \leftarrow \text{OT}_3(\text{OT}.cr$s$^i$, $\text{ots}_2$, $\omega'$),$$

Output $x$.

**Security.** We need to provide a Sim$^{\text{Com}}_\text{COMMIT}$ s.t. $\forall G \in \mathcal{G}^{\text{Com}}_\text{COMMIT}$ and $\forall Q \in \mathcal{Q}^{\text{Com}}_\text{COMMIT}$, there exists a reach-extractor $\mathcal{E}^{\text{Com}}_\text{COMMIT}$ w.r.t. $(\mathcal{G}^{\text{Com}}_\text{COMMIT}, \mathcal{P}^{\text{Com}}_\text{COMMIT})$ which satisfies Definition 4.

- $\mathcal{E}^{\text{Com}}_\text{COMMIT}$:
  * Let $(\mathcal{E}_1, \mathcal{E}_2)$ be the extractors from the binding property of $\text{Com}$. $\mathcal{E}^{\text{Com}}_\text{COMMIT}$ configures the cr$s$ as $(\text{cr}s$, st$\mathcal{E}_1$) $\leftarrow$ $\mathcal{E}_1(1^\kappa)$. It then runs $Q$ honestly and lets it interact with $G$.
  * As part of the interaction, $Q$ sends $c$ to $G$
  * $\mathcal{E}^{\text{Com}}_\text{COMMIT}$ extracts $x$ as $x$ $\leftarrow$ $\mathcal{E}_2($st$\mathcal{E}_1$, $c$), samples a relaxed program $\Pi_x \in \mathcal{P}^{\text{Com}}_\text{COMMIT}$ and outputs $\Pi_x$, $X^* = \{x\}$.
  * Sim$^{\text{Com}}_\text{COMMIT} \left( \pi_{c,\text{cr}s,\hat{\mu}}, I_{\text{COMMIT}}, \Pi_x, \{ x, \hat{\mu}(\text{REACH}_{\Pi_x}(x)) \} \right)$:
    * The secrets available to the simulator are $\{ \hat{\beta}(2, i, x_i) \}_{i \in [l]}$.
    * Parse cr$s$ as $\{ \text{OT}.cr$s$^i \}_{i \in [l]}$.
    * Parse $c$ as $\{ \text{ots}_1^i \}_{i \in [l]}$.
    * For every $i \in [l]$, do the following:
      $$\text{ots}_2^i \leftarrow \begin{cases} \text{OT}_2(\text{OT}.cr$s$^i, \text{ots}_1^i, \hat{\beta}(2, i, 0), \bot), & \text{if } x_i = 0, \\ \text{OT}_2(\text{OT}.cr$s$^i, \text{ots}_1^i, \bot, \hat{\beta}(2, i, 1)), & \text{otherwise}. \end{cases}$$
    * Output the program $\rho[\text{cr}s, c, \{ \text{ots}_2^i \}_{i \in [l]}]$ (as described in Figure 16).
The rest of the argument can be easily completed by considering a sequence of \( l \) hybrids\(^{19}\) and using the sender’s security of the underlying OT scheme for proving indistinguishability between two consecutive hybrids.

\[ \square \]

### R3PO scheme \( \mathcal{O}_{\text{Com}}^{\text{COMMIT}} \)

**Security Parameter:** Let \( \kappa \) be the security parameter.

**OT Scheme:** Let \( \mathcal{OT} = (\mathcal{OT}.\text{Setup}, \mathcal{OT}_1, \mathcal{OT}_2, \mathcal{OT}_3) \) be an OT scheme.

**Commitment Scheme:** Let \( \mathcal{Com} = (\mathcal{Com}.\text{Setup}, \mathcal{Com}.\text{Com}, \mathcal{Com}.\text{Open}) \) be a commitment opening scheme.

**Input:** Program \( \pi_{e,c} \in \mathcal{P}_{\text{COMMIT}} \), \( \hat{\mu}(\hat{\beta}) \in \hat{\mathcal{M}}_{\text{COMMIT},1} \) be a message function, and \( \mathcal{I}_{\text{COMMIT}} \) be the partition function defined previously.

\[
\mathcal{O}_{\text{COMMIT}}^{\text{COMMIT}} (\pi_{e,c}, \hat{\mu}(\hat{\beta}), \mathcal{I}_{\text{COMMIT}}) :
\]
- Parse \( c \) as \( \{\mathcal{OT}.\text{crs}_i\}_{i \in [l]} \).
- Parse \( e \) as \( \{\mathcal{OTS}_2\}_{i \in [l]} \).
- For every \( i \in [l] \), construct ciphertext \( \mathcal{OTS}_2 \leftarrow \mathcal{OT}_2(\mathcal{OT}.\text{crs}_i, \mathcal{OTS}_1(\mathcal{OT}.\text{crs}_i, \mathcal{OTS}_2, \hat{\beta}(2, i, 0), \hat{\beta}(2, i, 1)) \).
- Output a program \( \rho(crs, e, \{\mathcal{OTS}_2\}_{i \in [l]}) \) which does the following:
  * Takes as input \( d = \{\omega^i\}_{i \in [l]} \)
  * Parses \( c \) as \( \{\mathcal{OTS}_1\}_{i \in [l]} \)
  * Sets \( x = \mathcal{Com}.\text{Open}(\text{crs}, c, d) \)
  * For every \( i \in [l] \), sets \( m_i = \mathcal{OTS}_3(\mathcal{OT}.\text{crs}_i, \mathcal{OTS}_2, x, \omega^i) \)
  * Outputs \( \{m_i\}_{i \in [l]} \)

---

**C.2 Signature-Checking R3PO**

Let \( \mathcal{Sig} = (\text{gen}, \text{sign}, \text{verify}, \text{punct}, \text{psign}) \) be a puncturable signature scheme (Definition 18). We define a class of reactive programs, where the one-step transition corresponds to checking the signature of a fixed message on a fixed verification key. That is, the reactive program has hardwired a message \( m \) and a verification key \( \text{vk} \), and transitions from the start state to the state \( \sigma_m \) on input a signature \( \tau \) if \( \text{vk} \) verifies \( (m, \tau) \). All states \( \sigma_m \) are defined as being in the same partition. Note that, such a reactive program is trivially reach-restricted.

One can define a generator class (and thus the interaction) in one of two ways. We show that (surprisingly) the same obfuscation is a valid R3PO for both cases.

- The generator picks the signature key pair and specifies the verification key \( \text{vk} \). We discuss this in Appendix C.2.4, where we describe for the general case: \( Q \) specifies a target message prefix \( \text{mpre} \).

\(^{19}\)To elaborate further on this, we consider a hybrid Hybrid\(^j\) \((j \in \{0, 1, \ldots, l\})\) where the first \( j \) ciphertexts \( \{\mathcal{OTS}_2\}_{i \in [l], i \leq j} \) (in the description of the final program \( \rho \)) are generated as by \( \text{Sim}_{\text{Com}}^{\text{COMMIT}} \), while the remaining ones are generated as in \( \mathcal{O}_{\text{COMMIT}}^{\text{COMMIT}} \). It can be easily seen that the output of Hybrid\(^0\) is the same as that of \( \text{REAL}(G, Q) \), while that of Hybrid\(^l\) is the same as \( \text{IDEAL}(G, Q) \); where these experiments were defined in Definition 4.
- A semi-honest adversary \( Q \) specifies the verification key \( v_k \). We discuss this in Appendix C.2.5.

**Notation.** Throughout this section, we will write \((m)_t\) for a string \( m \) and integer \( t \) to denote the \( t \)-bit prefix of \( x \). We write \( \text{mpre} \preceq m \) to denote the predicate: \( \text{mpre} \) a prefix of \( m \).

### C.2.1 R3PO for One-time Signature-Checking

Let \( 1\text{-Sign} = (1\text{-Sig}.gen, 1\text{-Sig}.sign, 1\text{-Sig}.verify) \) be a one-time signature scheme and \( F = (F\text{.gen}, F\text{.punct}, F\text{.eval}) \) be a puncturable pseudorandom function family. We define a family of generators as follows.

**Program family and Generator class.** Each program in \( \mathcal{P}^{-\text{Sig}}_1 \) over the state space \( \Sigma_{1\text{-Sign}} = \Sigma_1 \cup \Sigma_2 \) (where, \( \Sigma_1 = \{\sigma^1_{vk} | vk \in VK\} \) and \( \Sigma_2 = \{\sigma^2_m | m \in M\} \)), is parameterized by a message prefix \( \text{mpre} \) and specified by a start state \( \sigma^1_{vk} \) and transition function:

\[
\pi_{\text{mpre}}(\sigma^1_{vk}, (m, \tau)) = \begin{cases} 
\sigma^2_{m} & \text{if } \text{mpre} \preceq m \text{ and } 1\text{-Sig}.\text{verify}(vk, \tau, m) = 1 \\
\sigma^1_{vk} & \text{otherwise.}
\end{cases}
\]

The transition function is defined as: \( I_{1\text{-Sign}}(\sigma) = 1 \) if \( \sigma \in \Sigma_1 \), else \( 2 \). Each generator in \( G^{-\text{Sig}}_1 \) is of the form \( H^{-\text{Sig}}_{1\text{-Sign}} \| Z \) for PPT \( Z \) and a fixed \( H^{-\text{Sig}}_{1\text{-Sign}} \) which interacts with any \( Z, Q \) as follows.

- \( H^{-\text{Sig}}_{1\text{-Sign}} \) samples a PRF key \( s \leftarrow F\text{.gen}(1^n) \). For the rest of the interaction, we denote \((sk^l, vk^l)\) as a one-time signature key pair sampled using randomness \( F\text{.eval}(s, i) \). It will also sign at most a single message with each \( sk^l \) (if multiple requests on the same message come, it sends the same signature).
- It accepts polynomial queries from \( Q \):
  - Verification key query: it accepts \( i \) from \( Q \), and sends \( vk^l \) to \( Q \).
  - Signature query: it accepts \((i, m^l)\) from \( Q \), and sends \( 1\text{-Sig}.\text{sign}(sk^l, m^l) \) to \( Q \).
- It receives a target index \( I \), target message \( m \) and prefix size \( t \) from \( Q \), and sends \((vk^l, 1\text{-Sig}.\text{sign}(sk^l, m)) \) to \( Q \). It sets \( \text{mpre} = (m)_t \).
- It receives a message function \( \hat{\mu}(\beta) \in \hat{\mathcal{M}}_{I_{1\text{-Sign}}, 0} \) from \( Z \), punctures the key \( s \) as \( s = F\text{.punct}(s, \text{pre}) \) on each prefix \( \text{pre} \) of \( I \) \((\text{pre} \preceq I)\), and halts with output \((\pi_{\text{mpre}}[\sigma^1_{vk^l}], \hat{\mu}(\beta), s) \).

**R3PO.** We now show that, for an adversary class \( Q^{-\text{Sig}}_{1\text{-Sign}} \) with all PPT \( Q \) and a relaxed program class same as \( \mathcal{P}^{-\text{Sig}}_1 \), a 1-Sig scheme and an R3PO for \( G^{-\text{Sig}}_1 \) can be constructed from any OTSE scheme. Recall that, the message output corresponding to a state \( \sigma_m \) in partition 2 is \( \hat{\mu}(\beta)(\sigma_m) = \{\beta(j, m, j)\}_{j \in |m|} \) (refer Definition 6).

### An R3PO scheme for One-Time Signature Checking

Let \( \kappa \) be the security parameter.

Let \( \text{OTSE} \) be an OTSE scheme \((\text{OTSE}.\text{gen}, \text{OTSE}.\text{sign}, \text{OTSE}.\text{verify}, \text{OTSE}.\text{enc}, \text{OTSE}.\text{dec}) \)

Let \( \text{SKE} \) be a symmetric-key encryption scheme \((\text{SKE}.\text{gen}, \text{SKE}.\text{enc}, \text{SKE}.\text{dec}) \)
We prove that the obfuscation scheme in Figure 17 is indeed a valid R3PO scheme w.r.t. Lemma 7.

**Proof:**

R3PO scheme w.r.t. Lemma 7.

The signature scheme $E$ now give an extractor $OTSE_Z,Q$ if

\[
\begin{align*}
\text{let } &\{\text{mpre}\} = t. \\
\text{if } &t = 0, \text{ set } k = 0; \text{ else, sample } k \leftarrow SKE\text{.gen}(1^*) \text{ and secret-share it as } k = s_1 + \ldots + s_t \\
\text{construct a SKE ciphertext that encrypts the labels for mpre; for each index } j \in [t], \text{ construct an OTSE ciphertext that encrypts the secret-share } s_j \text{ on target } (j, \text{ mpre}_j)
\end{align*}
\]

\[
ct \leftarrow SKE\text{.enc}(k, \{\tilde{\beta}(2, j, \text{ mpre}_j)\}_{j \in [t]}) \quad \{ct_j \leftarrow OTSE\text{.enc}(vk, j, \text{ mpre}_j, s_j)\}_{j \in [t]}
\]

- for each bit position $j \in (t, n]$, construct two OTSE ciphertexts that encrypt the labels $\tilde{\beta}(2, j, 0)$ and $\tilde{\beta}(2, j, 1)$ for bit position $j$ and target bit 0, 1 respectively (under the secret-shared encryption key)

\[
x_{j,0} \leftarrow SKE\text{.enc}(k, \tilde{\beta}(2, j, 0)) \quad x_{j,1} \leftarrow SKE\text{.enc}(k, \tilde{\beta}(2, j, 1))
\]

\[
ct_{j,0} \leftarrow OTSE\text{.enc}(vk, j, 0, x_{j,0}) \quad ct_{j,1} \leftarrow OTSE\text{.enc}(vk, j, 1, x_{j,1})
\]

- output a program $M [\{vk, ct, \{ct_j\}_{j \in [t]}: \{ct_{j,b}\}_{j \in [t], b \in \{0,1\}}]$, which does the following:
  - takes as input $(\tau, m)$.
  - decrypts each OTSE ciphertext as: $\forall j \in [t], \ x_j = OTSE\text{.dec}(vk, \tau, m, ct_j)$
  - construct $k = x_1 + \ldots + x_t$
  - decrypts the SKE ciphertext to get labels: $\{\ell_j\}_{j \in [t]} = SKE\text{.dec}(k, ct)$
  - decrypts each OTSE ciphertext as: $\forall j \in (t, n], \ y_j = OTSE\text{.dec}(vk, \tau, m, ct_{j,m_j})$
  - decrypts the SKE ciphertexts to get labels: $\forall j \in (t, n], \ \ell_j = SKE\text{.dec}(k, y_j)$
  - outputs $\{\ell_j\}_{j \in [n]}$.

**Fig. 17.** R3PO Obfuscation for one-time signature checking from OTSE.

**Lemma 7.** If there exists a semi-honest OTSE scheme, then there exists a signature scheme 1-Sig and an R3PO scheme w.r.t. $(\Sigma_{1-Sig}^1, Q_{1-Sig}^1, P_{1-Sig}^1)$.

**Proof:** We prove that the obfuscation scheme in Figure 17 is indeed a valid R3PO scheme w.r.t. $(\Sigma_{1-Sig}^1, Q_{1-Sig}^1, P_{1-Sig}^1)$. The signature scheme 1-Sig in the R3PO is simply the underlying OTSE scheme; that is, 1-Sig = (OTSE, gen, OTSE, sign, OTSE, verify). Recall that, the generator gives out at most a single signature on the signing key of each index position $i$.

Let $Z, Q$ be arbitrary parties and the output of the interaction $(H^1_{1-Sig} \parallel Z : Q)$ be $(\pi^{(a)}, \tilde{\mu}^{(b)}, a_G; a_Q)$. We now give an extractor $E$ and simulator $Sim$ for the scheme, so that it satisfies Definition 4:

- $E$ trivially extracts the verification key $v_k^{\ell}$ from the last step of the interaction: where $Q$ sends a target index $I$, message $m$ from $Q$; and $H^1_{1-Sig}$ replies with $(v_k^{\ell}, \sigma)$. $E$ outputs $\pi[v_k^{\ell}]$ and $X^* = \{(m, \sigma)\}$.

- $Sim$ takes as input $\pi^{(a)}, \mu^{(b)}, \Pi$ and $\{\tilde{\mu}^{(a)}\}_{\pi \in X^*}$ (simply the labels $\{\tilde{\beta}(2, j, m_j)\}_{j \in [n]}$ for the message $m$). It outputs a simulated obfuscation $\tilde{\rho}$, where the OTSE ciphertexts corresponding to the bits $1 - m_j$ for each bit position $j \in (t, n]$ are encryptions of 0:

\[
ct_{\tilde{\rho},1-m_j} \leftarrow OTSE\text{.enc}(vk, j, 1, 0)
\]

Indistinguishability follows via a reduction to the underlying OTSE security.

---

20 where, $\rho$ is a circuit with $vk$ and ciphertexts hardcoded into it
**Significance of the message prefix.** The reach-restriction and obfuscation simulation only relies on the generator giving out at most a single signature on the signing key $sk^I$ for target index $I$. Thus, even for an empty prefix, i.e., $t = 0$, the above is a valid obfuscation scheme, and the construction will correspondingly have SKE ciphertexts on the key $k = 0$. But, if the prefix is the full message, i.e., $t = n$ and $m_{pre} = m$, then the reach-restriction or simulation no longer relies on the generator. Indeed, the program will be reach-restricted even if the verification key $vk^I$ is specified by a semi-honest $Q$. We shall use this key idea in Appendix C.2.5 to give an R3PO for signature-checking where $Q$ picks the verification key.

### C.2.2 Towards R3PO for Signature-Checking

In the previous section, we described a one-time signature-checking R3PO, where the generator was picking a tree of one-time keys (via a PRF seed), but the reactive program was on a single verification key (and reach-restriction relied on giving out a single signature on its signing key). We now expand the scheme and the interaction, so that, a path in the tree corresponds to a message prefix $m_{pre}$, and the signature is now a sequence of one-time signatures along the path from root to $m_{pre}$ (where, each intermediate signature is on the verification keys of the child nodes). This allows us to have a single verification key $vk^e$ corresponding to the root node. Refer to Figure 18 for the construction of the puncturable signature scheme.

Towards constructing an obfuscation for signature-checking (as a one-step generator) in the library, we first construct an obfuscation for a slightly different reactive program family and generator class. Here, the transition is not one-step, but a sequence of transitions, where in each transition, it verifies the verification keys of the child nodes via a signature for them on the signing key of the current node. Finally, we will show that, this can essentially be squished into a single one-step transition, to get R3PO for signature-checking.

**Program family.** Let the prefix length be $t$ and state space be $\Sigma_{\text{SIGN}}^{1,\text{Sig}} = \bigcup_{i=1}^{t} (\Sigma_{(i,0)} \cup \Sigma_{(i,1)}) \cup \Sigma_{t+1}$; where: for $i \in [t]$,

$$\Sigma_{(i,0)} = \{\sigma_{vk}^{(i,0)} \mid vk \in VK\} \quad \Sigma_{(i,1)} = \{\sigma_{vk_0,vk_1}^{(i,1)} \mid vk_0, vk_1 \in VK\}$$

and $\Sigma_{t+1} = \{\sigma_{vk}^{t+1} \mid m \in M\}$. The partition function is defined as $I_{\text{SIGN}}(\sigma_{x}^{i,b}) = 2 \cdot (i - 1) + b$, for $\sigma_{x}^{i,b} \in \Sigma_{(i,b)}$.

Each program in $P_{\text{SIGN}}^{1,\text{Sig}}$, with start state $\sigma_{vk}^{(1,0)} \in \Sigma_{(1,0)}$, is parameterized by a fixed prefix $m_{pre}$ and is defined by the transition function:

$$\pi_{m_{pre}}(\sigma_{vk}^{(i,0)}, y, \tau) = \sigma_{y}^{(i,1)} \quad \text{if } i \neq t \text{ and } 1\text{-Sig.verify}(vk, y, \tau) = 1$$

$$\pi_{m_{pre}}(\sigma_{vk_0,vk_1}^{(i,1)}, \epsilon) = \sigma_{vk_0}^{(i+1,0)} \quad \text{where } b = vk_{m_{pre}}$$

$$\pi_{m_{pre}}(\sigma_{vk}^{(t,0)}, m, \tau) = \sigma_{vk}^{t+1} \quad \text{if } m_{pre} = (m)_i \text{ and } 1\text{-Sig.verify}(vk, m, \tau) = 1$$

This corresponds to a sequence of two-step transitions. For $i < t$, at a state $\sigma_{vk}^{(i,0)}$, it accepts a signature on $y$, and transitions to the state $\sigma_{y}^{(i,1)}$. Then, it interprets $y$ as the concatenation of two verification keys $vk_0, vk_1$ and does an epsilon transition to the state $\sigma_{vk_0}^{(i+1,0)}$ by selecting the key corresponding to the bit value of $m_{pre}$ at position $i$. Finally, at a state $\sigma_{vk}^{(t,0)}$, it accepts a message $m$, signature $\tau$ and transitions to the state $\sigma_{vk}^{t+1}$ if $\tau$ verifies and $m_{pre}$ is the $i$-bit prefix of $m$.

**Notation.** We denote $sk^i, vk^i$ to denote signature key pair sampled using randomness $F_{\text{eval}}(s, i)$. We define $\text{sign}^e(s, m)$ as a sequence of signatures along the path via $(m)_i$, i.e., $\{vk^i, \tau_i\}$, $vk^{(m)_i}$, $\tau$ for $i \in [t]$, where each $\tau_i = 1\text{-Sig.sign}(sk^i, vk^{(0}_{0}||vk^{(1}_{1}) \text{ and } \tau = 1\text{-Sig.sign}(sk^{(m)_i}, m)$. 

54
Generator class. Each generator in \( \mathcal{G}_{\text{SIGN}^*}^{1-\text{Sig}} \) is of the form \( H_{\text{SIGN}^*}^{1-\text{Sig}} \| Z \), for all PPT \( Z \), and a fixed \( H_{\text{SIGN}^*}^{1-\text{Sig}} \), that interacts with \( Z \) and \( Q \) as follows:

- It samples a PRF key \( s \rightarrow \text{F\-gen}(1^k) \).
- It sends the root verification key \( \hat{v}^k \) to \( Q \).
- It accepts polynomially many queries of messages \( m_j \in M \) from \( Q \) and responds with \( \text{sign}^*(s, m_j) \).
- It then accepts a target message \( m \) from \( Q \), verifies that \( Q \) had not queried any message with prefix \( \text{mpre} = (m)_r \) and punctures the PRF key on each prefix \( \text{pre} \) of \( \text{mpre} \) as \( \text{PRF\-punct}(s, \text{pre}) \).
- It finally accepts a message function \( \hat{\mu}(\hat{\beta}) \in \hat{M}_{\text{SIGN}^*} \) from \( Z \) and outputs \( \langle \pi_{\text{mpre}}[\tau_1^k], \hat{\mu}(\hat{\beta}), s \rangle \).

To give an R3PO obfuscation scheme for the above class, we first show that it decomposes to a library of one-step program obfuscation schemes. We then invoke Theorem 1 to construct an obfuscation for it from the library of one-step program obfuscation schemes.

Lemma 8. \((\mathcal{G}_{\text{SIGN}^*}^{1-\text{Sig}}, \mathcal{Q}_{\text{SIGN}^*}^{1-\text{Sig}}) \) decomposes into a library of one-time signatures \((\mathcal{G}_{1-\text{sign}}^{1-\text{Sig}}, \mathcal{Q}_{1-\text{sign}}^{1-\text{Sig}}) \) and epsilon-transitions \((\mathcal{G}_{e}^{\Sigma}, \mathcal{Q}_{e}^{\Sigma}) \) (where, \( \Sigma = \Sigma_{1-\text{Sig}}^{1-\text{Sig}} \)).

Proof: We show that \((\mathcal{G}_{\text{SIGN}^*}^{1-\text{Sig}}, \mathcal{Q}_{\text{SIGN}^*}^{1-\text{Sig}}) \) decomposes into

\[
\mathcal{L} = (\mathcal{L}_{(1,0)}, \mathcal{L}_{(1,1)}, \ldots, \mathcal{L}_{(t,0)})
\]

where

\[
\mathcal{L}_{(i,b)} = \begin{cases} 
(\mathcal{G}_{1-\text{sign}}^{1-\text{Sig}}, \mathcal{Q}_{1-\text{sign}}^{1-\text{Sig}}) & \text{if } b = 0, \\
(\mathcal{G}_{e}^{\Sigma}, \mathcal{Q}_{e}^{\Sigma}) & \text{else.}
\end{cases}
\]

We prove this by explicitly giving a \( J, W, Z \) that satisfy Definition 5.

- Case \( b = 0 \): At this partition, \( \mathcal{G}_{\text{SIGN}^*}^{1-\text{Sig}} \) decomposes to \( \mathcal{G}_{1-\text{sign}}^{1-\text{Sig}} \) (with the appropriate prefix and partition function, that we skip here; refer Appendix C.5). Let \( G \in \mathcal{G}_{\text{SIGN}^*}^{1-\text{Sig}} \) be a generator that interacts with \( Q \in Q \).
  - Defining \( W \): \( W \) runs \( Q \) in a white-box way and interacts with either \( J \) or \( H_{1-\text{sign}}^{1-\text{Sig}} \) via the bottom channel.
    * \( W \) sends query \( k = 0 \) on the bottom channel and gets back \( \hat{v}^k \).
    * For each query \( m_j \) that \( Q \) makes in the top channel, it does the following. For each \( k < \text{mpre}_j \), it queries indices \( k||0, k||1 \) on the bottom channel and gets back \( \hat{v}^k \), \( v_{k^0} \), \( v_{k^1} \); it then queries \( (k, \hat{v}^k || v_{k^1}) \) on the bottom channel and gets back \( \tau^j_k \). It finally queries \( (\text{mpre}_j, m_j) \) on the bottom channel and gets back \( \tau^j_{k+1} \).
    * Similarly, for the target message \( m, \) prefix \( \text{mpre} \).
    * \( W \) finally sends the target index \( I = (m)_i, \) target message \( \hat{v}^k || v_{k^1} \) if \( i < t \), else \( m \), and prefix length \( 0 \) if \( i < t \), else \( n \) (where \( |m| = n \)) on the bottom channel.
  - Defining \( J \): \( J \) runs \( G \) in a black-box way and interacts with \( W \) via the bottom channel.
    * \( G \) sends \( \hat{v}^k \) on the top channel. \( W \) sends query \( \epsilon \) on the bottom channel. \( J \) sends \( \hat{v}^k \) on the bottom channel.
    * For each query \( m_j \) that \( Q \) makes and \( G \) responds with \( \tau^j \) on the top channel, it does the following. It responds to the queries of \( W \) from \( \tau^j \).
    * Similarly, for the target message \( m, \) prefix \( \text{mpre} \).

\(^{21}\) that is, all prefixes \( k \) of \( m_j \) upto length \( t - 1 \)
* It finally samples \( \tilde{\mu}^{(\beta)} \in \tilde{\mathcal{M}}_{\text{sign}^{*},i} \), receives target index \( I \), target message \( m \) and prefix size \( x \); sets target prefix mpre as \( (m)_x \) and halts with output \( \tilde{\pi}_{\text{mpre}}[\sigma^{(i,0)}], \tilde{\mu}^{(\beta)} \).

- **Defining** \( Z \): \( Z \) runs \( H^{1-\text{sign}}_{1-\text{sig}} \) in a black-box way and interacts with \( Q \) via the top channel.
  - \( W \) queries \( \epsilon \) in the bottom channel and \( H^{1-\text{sign}}_{1-\text{sig}} \) responds with \( \epsilon \). \( Z \) sends \( \epsilon \) in the top channel.
  - For each query \( m^j \) that \( Q \) makes in the top channel, \( W \) queries a sequence of verification keys and sequence of messages; and \( H^{1-\text{sign}}_{1-\text{sig}} \) responds with corresponding keys and signatures. \( Z \) constructs the overall signature from it and sends in the top channel.
  - Similarly, for the target message \( m \), prefix mpre.
  - \( Z \) finally samples \( \tilde{\mu}^{(\beta)} \in \tilde{\mathcal{M}}_{\text{sign}^{*},0} \) and halts with output \( (\pi^{(\alpha)}, \mu^{(\beta)}) \).

- Case \( b = 1 \): At this partition, \( G^{1-\text{sign}}_{\text{sign}} \) decomposes to \( G^{1-\text{sign}}_{\text{sign}} \) (with the appropriate prefix and partition function, that we skip here; refer Appendix C.5). Let \( G \in G^{1-\text{sign}}_{\text{sign}} \) be a generator that interacts with \( Q \in Q \).

- **Defining** \( W \): \( W \) runs \( Q \) in a white-box way and interacts with either \( J \) or \( H^{1-\text{sign}}_{1-\text{sig}} \) via the bottom channel. \( W \) sees the interaction in the top channel. Let the target message be \( m \) and target prefix be mpre that \( Q \) sends in the top channel, and \( \tau \) be the response. Let \( I \) be the \( i - 1 \)-bit prefix of \( m \) and \( b = m_1 \); \( W \) extracts the keys \( \epsilon, \epsilon_1 \) from \( \tau \) and sends the states \( \sigma_1 = \sigma^{(i,1)}_{\epsilon, \epsilon_0, \epsilon_1, \epsilon}, \sigma_2 = \sigma^{(i+1,0)}_{\epsilon, \epsilon_1} \) in the bottom channel.

- **Defining** \( J \): \( J \) runs \( G \) in a black-box way and interacts with \( W \) via the bottom channel. It finally samples \( \tilde{\mu}^{(\beta)} \in \tilde{\mathcal{M}}_{\text{sign}^{*},i} \), receives receives the states \( \sigma_1, \sigma_2 \) from \( W \) and halts with output \( \pi_{\sigma_2}[\sigma_1], \tilde{\mu}^{(\beta)} \).

- **Defining** \( Z \): \( Z \) runs \( H^{1-\text{sign}}_{1-\text{sig}} \) in a black-box way and interacts with \( Q \) via the top channel. \( Z \) runs \( G \) internally, has it interact on the top channel, and outputs the output of \( G \).

The following corollary follows directly from Lemma 8 and Theorem 1.

**Corollary 1.** *If there exists a one-time signature primitive 1-Sig with an R3PO scheme w.r.t. \((G^{1-\text{sign}}_{\text{sign}}, Q^{1-\text{sign}}_{\text{sign}}, P^{1-\text{sign}}_{\text{sign}})\), then there exists an R3PO scheme w.r.t. \((G^{1-\text{sign}}_{\text{sign}}, Q^{1-\text{sign}}_{\text{sign}}, P^{1-\text{sign}}_{\text{sign}})\) for any message function class \( \mathcal{M} \).*

**C.2.3 R3PO for a Puncturable Signature Scheme**

All that is left is to squish the R3PO for \( G^{1-\text{sign}}_{\text{sign}} \) into a one-step transition and interpret as a R3PO for \( G^{1-\text{sign}}_{\text{sign}} \). We state this as the following lemma.

**Lemma 9.** *If there exists an one-time signature primitive 1-Sig with an R3PO scheme w.r.t. \( G^{1-\text{sign}}_{\text{sign}} \), then there exists a puncturable signature scheme \( \text{Sig} \) and an R3PO scheme w.r.t. \((G^{\text{Sig}, \mathcal{M}}_{\text{sign}}, Q^{\text{Sig}}, P^{\text{Sig}}_{\text{sign}})\) for any message function space \( \mathcal{M} \).*

*Proof:* \( \text{Sig} \) corresponds to the signature scheme that was implicit in the generator \( H^{1-\text{sign}}_{\text{sign}} \) above. We explicitly state it again in Figure 18. In the figure, the symbol \( \langle x \rangle_i \) denotes the \( i \)-bit long prefix of the string \( x \). We describe the wrapper on the R3PO of \( G^{1-\text{sign}}_{\text{sign}} \) in Figure 19.

**C.2.4 R3PO for Signature-Checking with Generator key**

In this section, we consider the first case, the generator samples the signature key pair. The target message from \( Q \) can be generalized to fixing only a prefix mpre and reach restriction holds so long as generator gives out at most a single signature for the target prefix mpre. We now describe for the general case.
Security Parameter: Let $\kappa$ be the security parameter.

Let $1$-Sig $= (1$-Sig.gen, $1$-Sig.sign, $1$-Sig.verify) be a one-time signature scheme.

Let $F = (F$.gen, $F$.punct, $F$.eval) be a puncturable pseudorandom function (PPRF) family.

$\text{Sig}.\text{gen}(1^\kappa) \rightarrow (\text{Sig}.vk, \text{Sig}.sk)$:

$$s \leftarrow F.\text{gen}(1^\kappa), (\text{Sig}.vk', 1$-Sig.sk') $\leftarrow 1$-Sig$.\text{gen}(1^\kappa; F.\text{eval}(s, \epsilon)).$$

Set $(\text{Sig}.vk, \text{Sig}.sk) := (1$-Sig.\text{vk}', s)$.

$\text{Sig}.\text{sign}(\text{Sig}.sk, x) \rightarrow \tau$: Parse $\text{Sig}.sk$ as $s$.

$$(1$-Sig.\text{vk}', 1$-Sig.\text{sk}') $\leftarrow 1$-Sig$.\text{gen}(1^\kappa; F.\text{eval}(s, \epsilon)).$$

$\forall i \in [t]:$

$$(1$-Sig.\text{vk}_0^{i+1}, 1$-Sig.\text{sk}_0^{i+1}) $\leftarrow 1$-Sig$.\text{gen}(1^\kappa; F.\text{eval}(s, (x), ||0)),$$

$$(1$-Sig.\text{vk}_i^{i+1}, 1$-Sig.\text{sk}_i^{i+1}) $\leftarrow 1$-Sig$.\text{gen}(1^\kappa; F.\text{eval}(s, (x), ||1)),$$

$$m_i = 1$-Sig.\text{vk}_0^{i+1}||1$-Sig.\text{vk}_i^{i+1},$$

$$\tau_i \leftarrow 1$-Sig$.\text{sign}(1$-Sig.\text{sk}_s^{i+1}, m_i).$$

Finally, do

$$\tau_x \leftarrow 1$-Sig$.\text{sign}(1$-Sig.\text{sk}_s^{x+1}, x).$$

Set $\tau := \left\{(\tau_i, m_i)\right\}_{i \in [t]}, (\tau_x, x)$.

$\text{Sig}.\text{verify}(\text{Sig}.vk, \tau, x) \rightarrow b$: Parse $\tau$ as $\left\{(\tau_i, m_i)\right\}_{i \in [t]}, (\tau_x, x)$.

$\forall i \in [t]:$

Parse $m_i$ as $1$-Sig.\text{vk}_0^{i+1}||1$-Sig.\text{vk}_i^{i+1},

$$b_i = 1$-Sig$.\text{verify}(1$-Sig.\text{vk}_s^{i+1}, \tau_i, m_i).$$

If each $b_i = 1$ and $1$-Sig$.\text{verify}(1$-Sig.\text{vk}_s^{x+1}, \tau_x, x) = 1$, then set $b := 1$, else $b := 0$.

$\text{Sig}.\text{punct}(\text{Sig}.sk, \text{mpre}) \rightarrow \text{Sig}.sk'$: Output $F.\text{punct}(s, \text{mpre})$.

$\text{Sig}.\text{psign}(\text{Sig}.sk', x) \rightarrow \tau'$: Output $\text{Sig}.\text{sign}(\text{Sig}.sk', x)$.

Fig. 18. Puncturable Signature from One-Time Signature

The Program class and Generator class. The generator class $\mathcal{G}_\text{SIGN}^\mathcal{M}$ is a set of generators of the form $\mathcal{H}_\text{SIGN}^{\mathcal{M}}$ for a fixed $\mathcal{H}_\text{SIGN}$ and all polynomial time $\mathcal{Z}$. The generator $\mathcal{H}_\text{SIGN}$ behaves as follows. It samples a signature key pair $(sk, vk)$ and sends $vk$ to $Q$. It then accepts polynomial queries $m_i \in \mathcal{M}$ from $Q$ and responds with $sign(sk, m_i)$. It then accepts a target message $m$ from $Q$, sets $\text{mpre}$ to be the $t$-bit prefix of $m$, verifies that at most a single signature with $\text{mpre}$ was given out and punctures the key as $\text{punct}(sk, \text{mpre})$. It finally accepts a message function $\mu^{(j)} \in \mathcal{M}$ from $Z$ and outputs a reactive program $(\mu^{(j)} \mid \pi_{\text{mpre}}[\sigma^1_{vk}], \mu^{(j)}, s)$ with start state $\sigma^1_{vk}$, whose transition function w.r.t. a state space $\Sigma = \{\sigma^1_{vk} \mid vk \in \mathcal{V}K, \mu^{(j)} \mid \sigma^0_{m} \mid m \in \mathcal{M} \}$ is
R3PO scheme $O^\text{Sig}_{\text{SIGN}}$

- Parse $vk$ as 1-Sig.$vk$.
- Sample $\hat{\mu}(\hat{\beta}) \in \hat{\mathcal{M}}_{\text{SIGN}} \times \sigma$.
- Set $\rho \leftarrow O^{1-\text{Sig}}_{\text{SIGN}}(\pi_{\text{mpre}}[\sigma^1_{vk}], \hat{\mu}(\hat{\beta}))$.
- Outputs a program $\rho'[\rho]$ as follows:
  - Takes as input $(\tau, x)$, where $\tau$ is parsed as $(\{\tau_i\}_{i \in [t]}, (\tau_x, x))$.
  - Traverse the program $\rho$ by sequentially providing input $\tau_i$ interleaved with $\epsilon$-transitions, for every $i \in [t]$.
  - Finally input $(\tau_x, x)$ and output $\rho'$s outputs.

### Fig. 19. Signature checking obfuscator

Specified as follows:

$$\pi_{\text{mpre}}(\sigma^1_{vk}, (\tau, m)) = \begin{cases} 
\sigma^2_m & \text{if verify}(vk, \tau, m) = 1 \text{ and mpre is a prefix of } m \\
\sigma^1_{vk} & \text{otherwise.} 
\end{cases}$$

The partition function is defined as: $I_{\text{SIGN}}(\sigma^1_{vk}) = 1$ for all $vk \in VK$ and $I_{\text{SIGN}}(\sigma^2_m) = 2$ for all $m \in M$.

### Adversary class and Reach-Extractor.

The adversary class is the set of all PPT $Q$. For any adversary $Q$, let the target message be $m$ and signature be $\tau$. There is a trivial extractor that simply outputs $\pi_{\text{mpre}}[\sigma^1_{vk}]$ and $X^* = \{(m, \tau)\}$ from the transcript of $Q$’s interaction.

### R3PO

We show that a puncturable signature scheme $\text{Sig}$ and an R3PO w.r.t. $(G^\text{Sig}, M^\text{SIGN}, Q^\text{SIGN}, P^\text{SIGN})$ can be built from any one-time signature with encryption (OTSE) scheme.

**Lemma 2 (Restated).** If there exists a semi-honest secure OTSE scheme, then there exists a puncturable signature scheme $\text{Sig}$ and an R3PO scheme $O^\text{Sig}_{\text{SIGN}}$ w.r.t. $(G^\text{Sig}_M, Q^\text{SIGN}_M, P^\text{SIGN}_M)$ for any message function space $\mathcal{M}$.

**Proof:** This result follows from Lemma 7, Corollary 1 and Lemma 9.

We prove this for the message function class $\hat{\mathcal{M}}$ (Definition 6). R3PO for any polynomial message function class can easily be derived by applying the composition theorem via a trivial decomposition to the same class but for $\hat{\mathcal{M}}$.

To build the required R3PO, we first build an R3PO for a one-time signature-checking scheme (Appendix C.2.1). We then build an R3PO for a signature-checking scheme, but with multiple transitions (Appendix C.2.2). We finally squish this to get the required construction.

**C.2.5 R3PO for Signature-Checking with Adversarial key**

Note that, the generalization to fix only the prefix mpre in the reactive program is not reach-restricted for an adversarial verification key, since the adversary can sign arbitrary messages and transition to all the corresponding states. Thus, we only consider reactive programs that have hardwired a message $m$ and a verification key $vk$, and transition from the start state to the state $\sigma_m$ on input a message $m$ and signature $\tau$ if $vk$ verifies $(\tau, m)$.
The Program class and Generator class. The generator class is a set of generators of the form \( H \parallel Z \) for a fixed \( H \) and all PPT \( Z \). The generator \( H \) behaves as follows. It accepts a verification key \( vk \) and a message \( m \) from \( Q \) and outputs a reactive program with start state \( \sigma^1_{vk} \), whose transition function w.r.t. a state space \( \Sigma = \{ \sigma^1_{vk} \mid v \in VK \} \cup \{ \sigma^2_m \mid m \in M \} \) is specified as follows:

\[
\pi_m(\sigma^1_{vk}, (\tau)) = \begin{cases} 
\sigma^2_m & \text{if verify}(vk, \tau, m) = 1 \\
\sigma^1_{vk} & \text{otherwise}.
\end{cases}
\]

The partition function is defined as: \( I_{\text{SIGN}}(\sigma^1_{vk}) = 1 \) for all \( vk \in VK \) and \( I_{\text{SIGN}}(\sigma^2_m) = 2 \) for all \( m \in M \).

Adversary class. The adversary class is a set of all PPT \( Q \) that honestly pick the signature key pair as \((sk, vk) \leftarrow \text{Sig}.\text{gen}(1^n)\) and sends \( vk \) to the generator.

R3PO. There exists a trivial obfuscation scheme for \( G^{\text{Sig}_{Q\text{SIGN}}} \) via a general epsilon transition (refer Appendix C.4).

Note that, in the construction of R3PO for \( G^{\text{1-Sig}_{Q\text{SIGN}}} \) from \( G^{\text{1-Sig}_{\text{Q}}} \) (Appendix C.2.2), we had a target prefix \( \text{mpre} \) in the reactive program, and the reach-restriction to prefix \( \text{mpre} \) was enforced via epsilon-transitions (where it transitions along the path to \( \text{mpre} \) and an explicit prefix check in the last one-step transition was not needed. The intention of decomposing at the last transition to \( G^{\text{1-Sig}_{\text{Q}}} \) with a target prefix \( \text{mpre} \) was to allow the same obfuscation to also be a valid R3PO for \( G^{\text{Sig}_{Q\text{SIGN}}} \). This follows from the fact that the R3PO for \( G^{\text{1-Sig}_{\text{Q}}} \) over a target prefix \( n \) (i.e., full message as target prefix) is secure even for an adversary that picks the verification key. We now prove the lemma.

**Lemma 10.** If there exists a semi-honest secure OTSE scheme, then there exists a puncturable signature scheme \( \text{Sig} \) and a PPT program \( O_{\text{Sig}} \) s.t. \( O_{\text{Sig}} \) is an R3PO obfuscation scheme w.r.t. \( (g^{\text{Sig}_{Q\text{SIGN}}}, Q, \tilde{P}^{\text{Sig}_{Q\text{SIGN}}}) \) as well as \( (g^{\text{Sig}_{\text{SIGN}-PO}}, Q, \tilde{P}^{\text{Sig}_{\text{SIGN}-PO}}) \).

**Proof:** We define \( g^{\text{Sig}_{\text{SIGN}-PO}} \) as a set of generators in \( g^{\text{Sig}_{\text{SIGN}}} \) that output reactive programs with full message as the target prefix, that is, \( |\text{mpre}| = n \). Such a generator decomposes as in **Lemma 8**; where in, it decomposes to \( G^{\text{1-Sig}_{\text{Q}}} \) at each partition \( (i, 0) \) with target prefix length \( 0 \) (that is, no target prefix) for each \( i < n \) and decomposes to \( G^{\text{1-Sig}_{\text{Q}}} \) at partition \( (n, 0) \) with prefix length \( n \) (that is, message \( m \) is the target prefix).

We now show that, for any adversary class of all semi-honest adversaries \( Q \) that honestly pick the signature key pair (as in Figure 18), the R3PO for \( G^{\text{Sig}_{Q\text{SIGN}-PO}} \) is also a valid R3PO for \( G^{\text{Sig}_{Q\text{SIGN}}} \) over the same relaxed program class. Formally,

- Extractor \( E \): For an adversary \( Q \), it trivially extracts the verification key \( vk \) and target message \( m \) from the transcript of the interaction between any \( G \in g^{\text{Sig}_{Q\text{SIGN}}} \) and \( Q \). It extracts the signature key \( sk \) from \( Q \), constructs the signature for \( m \) and outputs the same relaxed program as in \( g^{\text{Sig}_{\text{SIGN}-PO}} \).

- Simulator \( Sim \): It takes as input \( \pi(\cdot), \mu(\cdot), \Pi_m \) and \( \{ \mu(\beta(\text{REACH}_I(\pi))) \}_{\pi \in X^*} \) (simply the labels for \( m \)). It outputs a simulated obfuscation \( \tilde{\rho} \) as follows. It internally runs the obfuscator for \( g^{\text{Sig}_{\text{SIGN}-PO}} \) and outputs its output.

Reach-restriction and obfuscation security trivially follows from the corresponding guarantees of \( g^{\text{Sig}_{\text{SIGN}-PO}} \). This can be seen in the underlying construction, where at the last partition \( (n, 0) \), the obfuscator used the one-step obfuscation of \( G^{\text{1-Sig}_{\text{Q}}} \) with target prefix length \( n \) and target prefix \( m \). Thus, the only message the adversary can open to is \( m \). 

\( \square \)
C.3 Hash-Opening R3PO

Let $\text{Hash} = (\text{crsGen}, \text{hash}, \text{openBlock}, \text{acceptBlock})$ be a block-openable collision-resistant hash scheme (Definition 13), wherein the hash digest on a matrix $D$ can be opened via the openBlock algorithm to any index $i$ in $D$. We define a class of reactive programs, where the one-step transition corresponds to the opening of a fixed hash digest at a fixed index position $i$.

**The Program family and Generator class.** The generator class $G^{\text{Hash}}$ is a set of generators of the form $H \parallel Z$ for a fixed $H$ and all PPT $Z$, where $H$ behaves as follows. It accepts a setup $\text{crs}$, a hash digest and a block index $i$ from $Q$. It also accepts a message function $\mu^{(\beta)} \in M$ from $Z$. It then outputs a reactive program $(\pi_{\text{digest}, i, \text{crs}}, \tilde{\mu}^{(\beta)})$, whose transition function w.r.t. a state space $\Sigma = \{\text{start}\} \cup \{\sigma_y | y \in M\}$ is specified as follows:

$$\pi_{\text{digest}, i, \text{crs}}(\text{start}, k) = \begin{cases} 
\sigma_y & \text{if } \text{acceptBlock}(\text{crs, digest, } i, k) = y \\
\text{start} & \text{otherwise}.
\end{cases}$$

The partition function is defined as: $I_\text{Hash}(\text{start}) = 1$ and $I_\text{Hash}(\sigma_y) = 2$ for all $y \in M$.

**Adversary class and Reach-Extractor.** The adversary class is defined as the set of all PPT $Q^T$, where $T$ is a setup in $\{\text{crsGen}\}$ that samples the crs and each $Q$ honestly constructs the hash of a message $D \in M$ as digest $= \text{hash}(\text{crs, } y)$. For any adversary $Q$, there exists a trivial extractor that simply looks inside $Q$, extracts the message $D$, digest digest and the opening $k$ for target index $i$; and outputs $\pi_{\text{digest}, i, \text{crs}}$, $X^* = \{k\}$.

**R3PO.** We first build a hash scheme with factor-2 compression (that is, a matrix $D$ of 2 blocks is hashed into a digest of single block) and a R3PO for it that supports one bit of indexing from laconic OT with factor-2 compression (Appendix C.3.1). We then do a domain extension of the hash scheme to get a hash scheme that supports arbitrary compression and give a R3PO for hash opening w.r.t. this hash scheme, but with multiple transitions (Appendix C.3.2). We finally squish it to get the required construction of an arbitrarily-compressing hash scheme $\text{Hash}$ and an R3PO w.r.t. $(G^{\text{Hash}}, Q^{\text{Hash}}, \tilde{P}^{\text{Hash}})$ (please refer to Appendix C.3.3).

C.3.1 Block-Openable CRH with Factor-2 Compression

We now show how to construct a hash scheme with factor-2 compression Hash and an R3PO scheme w.r.t. $(G^{\text{Hash}}, Q^{\text{Hash}}, \tilde{P}^{\text{Hash}})$ from laconic OT scheme with factor-2 compression.

---

**Fig. 20.** Block-Openable CRH with factor-2 compression from Laconic OT with factor-2 compression

**Lemma 11.** Assuming a laconic OT scheme with factor-2 compression, there exists a block-openable collision-resistant hash scheme with factor-2 compression.

---

For a hash scheme with factor-2 compression the block index is simply a bit, since there are only two blocks.
Proof: We show that the scheme in Figure 20 is a block-openable CRH with factor-2 compression. Correctness of the scheme follows from the fact that $\ell OT$.hash is a deterministic function. The factor-2 compression also follows from the factor-2 compression of $\ell OT$. Collision-resistance of Hash follows from sender privacy of $\ell OT$ in the following way. For the sake of contradiction, assume that there exists an adversary Adv who is able to produce $(m, b, w)$ on input $\text{crs} \leftarrow \text{Hash}$.crsGen($1^k$) s.t.

$$\text{acceptBlock}(\text{crs}, \text{hash}(\text{crs}, m), b, w) \notin \{\bot, m_b\}$$

with some non-negligible probability, where $m = m_0||m_1$. Then there exists an adversary Adv$'$ and a distinguisher $D$ that can break the sender privacy property of $\ell OT$ as follows:

1. It gets $\text{crs}$ from the $\ell OT$ experiment and provides it to Adv.
2. It accepts $(m, b, w)$ from Adv. If $m_b \neq w_b$ (where $w = w_0||w_1$), then it chooses $L$ s.t. $m[L] \neq w[L]$ where $x[L]$ represents the bit at the $L$th location in the string $x$. If $m_b = w_b$, then it chooses any random $L$. Provide $m$ as $D$, location as $L$, two different messages $p_0$ and $p_1$ (does not matter if randomly chosen or not), and $\text{aux} = (\text{crs}, m, w, L, p_0, p_1)$ to the $\ell OT$ experiment.
3. $D$ gets $(\text{aux} = (\text{crs}, m, w, L, p_0, p_1), \text{ct})$ from the $\ell OT$ experiment. It does the following computation:

$$p^0 := \text{Receive}^m(\text{crs}, \text{ct}, L), p^1 := \text{Receive}^w(\text{crs}, \text{ct}, L).$$

If $p^0 = p_{m[L]}$, $p^1 = p_{w[L]}$ and $m[L] \neq w[L]$, then it outputs 0. Otherwise, it outputs 1.

Whenever Adv succeeds in breaking the privacy of Hash, it must be able to output a tuple $(m, b, w)$ s.t. $m_b \neq w_b$ and $\text{hash}(\text{crs}, m) = \text{hash}(\text{crs}, w)$. This follows from the description of Hash in Figure 20. In such a scenario, $(\text{Adv}', D)$ would be able to break the $\ell OT$ sender security experiment because the distribution created using the actual $\ell OT$.Send would contain both messages $p_0$ and $p_1$, while that created using $\ell OT$.Sim would contain only one of these messages. Since this scenario occurs with non-negligible probability, it follows that $(\text{Adv}', D)$ have a non-negligible advantage in the $\ell OT$ experiment. □

Lemma 12. Assuming a laconic OT scheme with factor-2 compression, there exists a block-openable CRH with factor-2 compression Hash and a R3PO scheme w.r.t. $(\mathcal{O}^H_{\text{Hash}}, \mathcal{Q}^H_{\text{Hash}}, \mathcal{P}^H_{\text{Hash}})$.

Proof: We instantiate Hash with the scheme in Figure 20. We now show that the scheme in Figure 21 is a R3PO scheme w.r.t. $(\mathcal{O}^H_{\text{Hash}}, \mathcal{Q}^H_{\text{Hash}}, \mathcal{P}^H_{\text{Hash}})$. Correctness of $\mathcal{O}^H_{\text{Hash}}$ follows from that of $\ell OT$. For proving security, we will have to describe extractor $E$ and simulator $\text{Sim}$ for the scheme, so that it satisfies Definition 4. Let $Z, Q$ be arbitrary parties s.t. the interaction between $H^H_{\text{Hash}}, Z$ and $Q$ follows the description of generator class $G^H_{\text{Hash}}$ in Appendix C.3. Note that we are restricting the adversaries to be semi-honest i.e., $Q$ behaves honestly w.r.t. some input database $D$ being used to compute digest. Therefore, the extractor can simply extract $D$ and output $(\pi_{\text{digest}, b, \text{crs}}, X^* = \{D\})$, where $(\text{digest}, b)$ is sent by $Q$ to $H^H_{\text{Hash}}$ in the interaction. The simulator $\text{Sim}$ behaves as follows:

1. Takes as input $(\pi_{\text{digest}, b, \text{crs}}, \hat{\mu}(\cdot), \pi_{\text{digest}, b, \text{crs}}, \{x, \hat{\mu}(\cdot)(\text{REACH}_{\pi_{\text{digest}, b, \text{crs}}}(x))\}_{x \in X^*})$.
2. Parse $x = D$ in $X^*$.
3. Let $t = |\text{digest}|$.
4. If $b = 0$, then $t_0 := 0$ else $t_0 := t$.
5. For each bit position $j \in [t]$, construct a single $\ell OT$ ciphertext as follows

$$\text{ct}_j \leftarrow \ell OT.\text{Sim}(\text{crs}, D, j + t_0, \hat{\beta}(2, j, D[j + t_0])).$$

61
An R3PO scheme for Hash Opening with factor-2 compression

Let $\kappa$ be the security parameter.
Let $\ellOT = (\text{crsGen}, \text{hash}, \text{Send}, \text{Receive})$ be a laconic OT scheme with factor-2 compression.
Let $\text{Hash} = (\text{crsGen}, \text{hash}, \text{openBlock}, \text{acceptBlock})$ be a block-openable CRH scheme with factor-2 compression (such as in Figure 20).

**Obfuscator** $G_{\text{Hash}}^{\text{Hash}}(\pi_{\text{digest}}, b, crs, \mu^{\hat{\beta}})$:
- Let $t = |\text{digest}|$.
- If $b = 0$, then $t_0 := 0$ else $t_0 := t$.
- For each bit position $j \in [t]$, construct a $\ellOT$ ciphertext that encrypts the labels $\hat{\beta}(2, j, \cdot)$ for bit position $j$ as follows:
  $ct_j \leftarrow \ellOT.\text{Send}(crs, \text{digest}, j + t_0, \hat{\beta}(2, j, 0), \hat{\beta}(2, j, 1))$.
- Output $\rho[crs, b, \{ct_j\}_{j \in [t]}]$.

**Evaluating an obfuscation** $\rho[crs, b, \{ct_j\}_{j \in [t]}]$ on input $k$:
- Parse $k$ as $D$.
- Let $t = |D|/2$.
- If $b = 0$, then $t_0 := 0$ else $t_0 := t$.
- For each $j \in [t]$, decrypt $\ellOT$ ciphertext as follows:
  $m_j := \ellOT.\text{Receive}^D(crs, ct_j, j + t_0)$.
- Output $\{m_j\}_{j \in [t]}$.

**Fig. 21.** R3PO for Hash Opening for block-openable CRH with factor-2 compression

6. Output $\rho[crs, b, \{ct_j\}_{j \in [t]}]$.

Indistinguishability between the distributions $\text{REAL}(G_{\text{Hash}}^{\text{Hash}}, Q)$ and $\text{IDEAL}(G_{\text{Hash}}^{\text{Hash}}, Q)$ can be shown by a sequence of $t$ hybrids, where $t = |D|/2 = |\text{digest}|$. In the $i^{th}$ hybrid for every $i \in [t]$, we change how the ciphertext $ct_i$ is being generated inside the output program $\rho[crs, b, \{ct_j\}_{j \in [t]}]$. While in the $(i - 1)^{th}$ hybrid $ct_i$ is obtained by the output of $\ellOT.\text{Send}$ with the other message being $\hat{\beta}(2, i, 1 - D[i + t_0])$, in the $i^{th}$ hybrid it is chosen to be the output of $\ellOT.\text{Sim}$ with only $\hat{\beta}(2, i, D[i + t_0])$ provided. Proving indistinguishability between these 2 hybrids reduces to sender security of $\ellOT$. Note that the first hybrid is the same as $\text{REAL}$ while the final hybrid is same as $\text{IDEAL}$. □

**C.3.2 Towards R3PO for Hash Opening: Domain Extension**

In this section, we show how to construct a block-openable CRH scheme with arbitrary compression and an R3PO for its program family from a R3PO of a hash-opening with factor-2 compression. Consider the following description of program classes that corresponds to hash-opening with arbitrary compression but with multiple transitions.

- **Program family $P_{\text{Hash}}^{\text{Hash}*}$**: This family is specified by a block-openable CRH with factor-2 compression $\text{Hash}$. Each program in the program family is parameterized by a hash output digest, a location $L$, and a common random string $crs$. Let the location $L$ be specified as the sequence $(b_1, b_2, \ldots, b_l)$ in binary. A
Corollary 2. If there exists a block-openable CRH with factor-2 compression \( \text{Hash} \), then there exists an R3PO scheme w.r.t. \( (G_{\text{Hash}}^*, Q_{\text{Hash}}^*, \hat{p}_{\text{Hash}}^*) \).
C.3.3 R3PO for Hash Opening

We first note that a block-openable CRH with arbitrary compression can be built from any block-openable CRH with factor-2 compression. This is essentially the merkle-tree based domain extension and is similar to the description of the generator \( \mathcal{P}_{\text{Hash}}^{\star} \) above.

**Lemma 14.** If there exists a block-openable CRH with factor-2 compression \( \text{Hash}' \), then there exists a block-openable CRH with arbitrary compression \( \text{Hash} \).

**Proof:** The description of Hash from \( \text{Hash}' \) has been given in Figure 22. For ease of exposition, assume that \( n = 2^d \) for some \( d \in \mathbb{Z} \), and input messages consist of \( n \) blocks of \( \kappa \) length each. While Hash can be instantiated using a Merkle tree with \( \text{Hash}' \) being the underlying hash scheme, openBlock w.r.t. a particular location \( L \) corresponds to the sequence of children (two for each node) in the tree along the path given by \( L \). acceptBlock then corresponds to just checking the hashes along this path only and then outputting the correct leaf. Roughly speaking, collision resistance of Hash follows because finding a collision in the Merkle tree would correspond to finding a collision in \( \text{Hash}' \) at the node where the two sequences diverge. We skip mentioning the full details here as this is a standard construction.

<table>
<thead>
<tr>
<th>Let Hash’ = (crsGen, hash, openBlock, acceptBlock) be a doubly-compressing hash scheme.</th>
</tr>
</thead>
<tbody>
<tr>
<td>• crsGen(1(^n)) : Run crs ← Hash’.crsGen(1(^n)) and output crs.</td>
</tr>
<tr>
<td>• hash(crs, m) : Let ( m = m_1 | \ldots | m_{n} ) s.t. (</td>
</tr>
<tr>
<td>• openBlock(crs, m, i) : Consider the Merkle tree with ( m ) forming the leaves like before. Output all the children (two for each node) in this tree along the path from the root digest to the leaf ( m_i ).</td>
</tr>
<tr>
<td>• acceptBlock(crs, digest, i, k) : Parse ( k ) as ( k_1, \ldots, k_d ), where ( d ) is the depth of the Merkle tree. Check if ( k_{j,b_{j}} = \text{Hash}'.\text{hash(crs, } k_{j+1}) ), for all ( j \in [d] ), where ( b_j ) denotes the child chosen at depth ( j ) to reach the ( i )th leaf.</td>
</tr>
</tbody>
</table>

**Fig. 22.** Arbitrarily-Compressing Hash from Doubly-Compressing Hash

**Lemma 15.** If there exists a laconic OT scheme with factor-2 compression, then there exists an arbitrarily-compressing hash scheme \( \text{Hash} \) and an R3PO scheme w.r.t. \( (\mathcal{O}_{\text{Hash}}, \mathcal{Q}_{\text{Hash}}, \mathcal{P}_{\text{Hash}}) \).

**Proof:** We instantiate the block-openable CRH with arbitrary compression \( \text{Hash} \) with the scheme in Figure 22. To get an R3PO scheme \( \mathcal{O}_{\text{Hash}} \) w.r.t. \( (\mathcal{G}_{\text{Hash}}, \mathcal{Q}_{\text{Hash}}, \mathcal{P}_{\text{Hash}}) \), we must put a wrapper around the R3PO scheme \( \mathcal{O}_{\text{Hash}}^{\star} \) w.r.t. \( (\mathcal{G}_{\text{Hash}}^{\star}, \mathcal{Q}_{\text{Hash}}^{\star}, \mathcal{P}_{\text{Hash}}^{\star}) \), as obtained in Corollary 2. Consider the description given in Figure 23.

Then we prove the following properties:

- **Correctness.** This follows from the description of \( \text{Hash} \), sampling of \( \beta \) and correctness of \( \mathcal{O}_{\text{Hash}}^{\star} \).
- **Efficiency.** This follows from the efficiency of \( \mathcal{O}_{\text{Hash}}^{\star} \).
- **Security.** We need to provide a \( \text{Sim}_{\text{Hash}}^{\star} \) s.t. \( \forall G \in \mathcal{G}_{\text{Hash}}^{\star} \) and \( \forall Q \in \mathcal{Q}_{\text{Hash}}^{\star} \), there exists a reach-extractor \( \mathcal{E}_{\text{Hash}} \) w.r.t. \( (\mathcal{G}_{\text{Hash}}, \mathcal{P}_{\text{Hash}}) \) which satisfies Definition 4. We describe these algorithms as follows:
R3PO scheme \(\mathcal{C}_{\text{Hash}}^{\text{Hash}}\)

\(\mathcal{C}_{\text{Hash}}^{\text{Hash}}(\pi_{\text{digest}}, i, \text{crs}, \hat{\mu}(\hat{\beta})) :\)
- Consider the program \(\pi'_{\text{digest}}, i, \text{crs} \in \mathcal{P}_{\text{Hash}}^{i}\) as described in Appendix C.3.2.
- Consider the new message function \(\hat{\beta}' \) as follows:
  \[
  \hat{\beta}'(i', j, b) = \begin{cases} 
  \hat{\beta}(2, j, b) & \text{if } i' = d, \\
  \{0, 1\}^\kappa & \text{otherwise},
  \end{cases}
  \]
  where \(d\) denotes the number of partitions in \(\pi'_{\text{digest}}, i, \text{crs}\).  
- Compute \(\rho \leftarrow \mathcal{C}_{\text{Hash}}^{i}(\pi_{\text{digest}}, i, \text{crs}, \hat{\mu}(\hat{\beta}')).\)
- Outputs a program \(\rho' [\rho]\) as follows:
  * Takes as input \(k = (k_1, \ldots, k_d)\).
  * Execute \(\rho\) on \(k_j\), for every \(j \in [d]\) in sequence.
  * Output the final output of \(\rho\).

Fig. 23. Hash Opening Obfuscator

- \(\mathcal{E}_{\text{Hash}}:\)
  * Since the interaction between \(G\) and \(Q\) is the same as that in the \(\text{HASH}^*\) generator class, \(\mathcal{E}_{\text{Hash}}\) could just invoke that extractor \(\hat{\mu}(\hat{\beta})\) to learn an input sequence \((k_1, \ldots, k_d)\), each \(k_i \in \{0, 1\}^{2\kappa}\). Then set \(X^* = \{k\}\) and output the relaxed program as \(\pi_{\text{digest}}, i, \text{crs} \in \mathcal{P}_{\text{Hash}}^{i}\) where \((\text{digest}, i, \text{crs})\) are obtained from the interaction.
  * \(\text{Sim}_{\text{Hash}}^{i}\left(\pi_{\text{digest}}, i, \text{crs}, \hat{\mu}(\hat{\beta}), \pi_{\text{digest}}, i, \text{crs}, \{x, \hat{\mu}(\hat{\beta})\} (\text{REACH}_{\pi_{\text{digest}}, i, \text{crs}}(x)) \} x \in X^*\) :
    * Parse \(x = k = (k_1, \ldots, k_d)\), and set \(X\) as the sequence \(k\) only.
    * Obtain \(\rho \leftarrow \text{Sim}_{\text{Hash}}^{i}(\pi'_{\text{digest}}, i, \text{crs}, \hat{\mu}(\hat{\beta}), \pi'_{\text{digest}}, i, \text{crs}, \{x, \hat{\mu}(\hat{\beta})\} (\text{REACH}_{\pi'_{\text{digest}}, i, \text{crs}}(x)) \} x \in X\), where \(\pi'_{\text{digest}}, i, \text{crs} \in \mathcal{P}_{\text{Hash}}^{i}\) and \(\hat{\beta}'\) is generated from \(\hat{\beta}\) as in Figure 23.
    * Output \(\rho' [\rho]\), where \(\rho'\) was defined in Figure 23.

Security of the above obfuscation scheme follows directly from that of \(\mathcal{C}_{\text{Hash}}^{i}\) as the interaction between the parties and the input sequences follow the same semantics in both classes. We skip those details here for brevity.

\[\square\]

C.4 Epsilon-Transition R3PO

The generator class \(G_{\Sigma}^{\Sigma}\) over state space \(\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3\) is a set of generators of the form \(H\|Z\), for a fixed \(H\) and all PPT \(Z\), where \(H\) behaves as follows. It accepts two state \(\sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2\) from \(Q\) and outputs a reactive program that transitions from \(\sigma_1\) (as the start state) to \(\sigma_2\) on input \(\epsilon\). That is:
\[
\pi_{\sigma_2}(\sigma_1, \epsilon) = \sigma^2
\]
This can be generalized to allow \(Q\) to specify a function \(f\). Then, the transition corresponds to transitioning from \(\sigma_1\) to \(f(\sigma_1) \in \Sigma_2\) on input \(\epsilon\). The corresponding partition function is simply \(I_i(\sigma) = i\) if \(\sigma \in \Sigma_i\).
An R3PO obfuscation scheme for Epsilon Transition. There is a trivial R3PO obfuscation scheme for $\mathcal{P}\Sigma_\epsilon$ w.r.t. adversary class $Q$ (all PPT $Q$) and a relaxed program class same as $\mathcal{P}\Sigma_\epsilon$):

$$O_\epsilon(\pi_f, \hat{\mu}(\beta)) = \hat{\mu}(\beta)(f(\sigma^1))$$

C.5 Generalizing the State Space and Partition Function

Let $G$ be a $(\mathcal{P}, \hat{\mathcal{M}}, \mathcal{I})$-generator class of the form $H\parallel Z$, for a fixed $H$ and PPT $Z$ (where $\mathcal{P}$ is defined w.r.t. a state space $\Sigma$ and $\mathcal{I} : |\Sigma| \rightarrow [N]$ is a partition function), and $O$ be an R3PO obfuscation for $(G, Q, \hat{\mathcal{P}})$. Let $P$ be a set of prefixes, $\Sigma' = P | | \Sigma^2$ be a state space, and $\mathcal{I}' : |\Sigma'| \rightarrow [N']$ be a partition function s.t. there exists an injective map $\eta : [N] \rightarrow [N']$ which satisfies

$$\forall \text{pre} \in P, \forall \sigma \in \Sigma, \mathcal{I}'(\text{pre} | | \sigma) = \eta(\mathcal{I}(\sigma))$$

We define the following wrapper programs. In particular, for all $\text{pre} \in P$, $\pi^{(\alpha)} \in \mathcal{P}$, we describe the program $\text{WrapOne}(\text{pre}, \pi^{(\alpha)})$ in Figure 25.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig24}
\caption{Transition Program $\pi_f$}
\end{figure}

\begin{itemize}
  \item Input: $(\sigma', x)$.
  \item Set $y$ as
    \begin{align*}
      y = \begin{cases} 
        \text{pre} | | \pi^{(\alpha)}(\sigma, x) & \text{if } \exists \sigma \text{ s.t. } \sigma^1 = \text{pre} | | \sigma, \\
        \sigma' & \text{otherwise.}
      \end{cases}
    \end{align*}
  \item Output $y$.
\end{itemize}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig25}
\caption{Wrapper Program $\text{WrapOne}(\text{pre}, \pi^{(\alpha)})$}
\end{figure}

Similarly, for all $\text{pre} \in P$, $\hat{\mu}(\beta) \in \hat{\mathcal{M}}$, we define the program $\text{WrapTwo}(\text{pre}, \hat{\mu}(\beta), \hat{\mu}(\beta))'$ in Figure 26.

Then, we modify $G$ to get a generator class $G'$ that supports state space $\Sigma'$ and partition function $\mathcal{I}'$ as follows:

- Program family $\mathcal{P}'$: Every program in $\mathcal{P}'$ is defined as a wrapper on a corresponding program $\pi^{(\alpha)} \in \mathcal{P}$.
  That is,
  \begin{equation*}
  \mathcal{P}' = \{ \text{WrapOne}(\text{pre}, \pi^{(\alpha)}) \mid \text{pre} \in P, \pi^{(\alpha)} \in \mathcal{P} \}
  \end{equation*}
- Program family $\hat{\mathcal{P}}'$: This program family is same as $\mathcal{P}'$.

\footnote{We basically consider all states of the form $\{\text{pre} | | \sigma\}_{\text{pre} \in P, \sigma \in \Sigma}$.}
Lemma 16. Let $G$ be a $(\mathcal{P}, \hat{M}, \mathcal{I})$-generator class and $O$ be an R3PO obfuscation for $(G, Q, \hat{P})$. Then for all prefixes $pre \in P$, partition function $\mathcal{I}'$ and injective map $\eta$ s.t. $\forall \sigma \in \Sigma, \eta(\mathcal{I}(\sigma)) = \mathcal{I}'(\text{pre}||\sigma)$, there exists an R3PO scheme for $(G', Q, \hat{P}')$.

Proof: To show that the scheme given in Figure 27 is a valid obfuscation scheme for $(G', Q, \hat{P}')$, we need to define a simulator $\text{Sim}'$ and an extractor $\mathcal{E}'$ satisfying our R3PO definition. Since $O$ is a valid obfuscation scheme for $(G, Q, \hat{P})$, there must exist a simulator $\text{Sim}$ and extractor $\mathcal{E}$ for security. $\mathcal{E}'$ can simply be defined to use $\mathcal{E}$ and $\text{Sim}'$ can also be defined to use $\text{Sim}$ for achieving our notion as follows. Note that the new message function has been split into two parts - one for the prefix and another for the original program. Since the prefix is known to the extractor, extraction is trivial as transitions only happen according to the original program. The first part of the message function, corresponding to the prefix, is unchanged for the entire program. The original simulator can be used for the second part. 

\[\blacksquare\]
D Details Omitted from Section 6: IBFE

Lemma 3 (Restated). If there exists a puncturable signature scheme $\text{Sig}$ and an R3PO scheme w.r.t. $(G_{\text{SIGN}}^{\text{Sig}}, Q_{\text{SIGN}}^{\text{Sig}}, P_{\text{SIGN}}^{\text{Sig}})$, then there exists a secure IBFE scheme.

Proof: We prove that the scheme in Figure 5 is a secure IBFE scheme satisfying Definition 7. It uses an R3PO scheme $G_{\text{SIGN}}^{\text{Sig}}$ w.r.t. $(G_{\text{SIGN}}^{\text{Sig}}, Q_{\text{SIGN}}^{\text{Sig}}, P_{\text{SIGN}}^{\text{Sig}})$ as described in Appendix C.2.4 (note that a generator in $G_{\text{SIGN}}^{\text{Sig}}$ is denoted as $H_{\text{SIGN}}^{\text{Sig}}$ there).

Let $\text{Adv} = (A_1, A_2)$ be an adversary for the IBFE security experiment in Figure 4 with advantage $\alpha$. We give a reduction to the R3PO security (Definition 4) of $G_{\text{SIGN}}^{\text{Sig}}$ by constructing a generator $G \in G_{\text{SIGN}}^{\text{Sig}}$, an adversary $Q \in Q_{\text{SIGN}}^{\text{Sig}}$ and a PPT distinguisher $D'$. Recall that, $G$ is of the form $H_{\text{SIGN}}^{\text{Sig}} \| Z$ for a fixed $H_{\text{SIGN}}^{\text{Sig}}$ (in the library) and PPT $Z$. We now describe $Z$, $Q$, $D'$ and the interaction:

- $Q$ internally runs $A_1$ in a straightline black-box way.
- $Q$ receives $\text{Sig}.\text{vk}$ from $H_{\text{SIGN}}^{\text{Sig}}$ and forwards it to $A_1$.
- KeyGen queries: $Q$ receives polynomial many key queries from $A_1$ of the form $(\text{id}_i, f_i)$, for each such query $Q$ sends query for signature of $(\text{id}_i \| f_i)$ to $H_{\text{SIGN}}^{\text{Sig}}$ and forwards the signature response to $A_1$.
- Challenge: $Q$ receives the challenge $(\text{id}^*, m_0, m_1)$ from $A_1$ and forwards it to $Z$. $Z$ sends $\text{id}^*$ as the target mpre to $H_{\text{SIGN}}^{\text{Sig}}$.
- $Q$ receives the punctured key sk (punctured on all signature queries above and on $\text{id}^*$) from $H_{\text{SIGN}}^{\text{Sig}}$.
- $Z$ samples $b \leftarrow \{0, 1\}$, sets $m := m_b$ and sends $\rho_{\text{id}^*,f}^{(m)}$ to $H_{\text{SIGN}}^{\text{Sig}}$ as the message function.
- $Z$ outputs $a_G = b$ as its aux output, $Q$ outputs $a_Q = (\text{id}^*, \text{sk}, m_0, m_1)$ as its aux output.

At the end of the interaction, $D'$ takes the output of the execution $(\rho, a_G, a_Q)$ as its input; where $\rho$ is the real obfuscation for the execution $\text{REAL}(G, Q)$ and the simulated obfuscation output by $\text{Sim}$ for the ideal execution $\text{IDEAL}(G, Q, E)$. $D'$ internally runs $A_2$ in a straightline black-box way and behaves as follows.

- $D'$ parses $a_G$ as $b$, parses $a_Q$ as $(\text{id}^*, \text{sk}, m_0, m_1)$ and sends $\rho$ to $A_2$ as the challenge ciphertext.
- KeyGen queries: $D'$ responds to polynomial many keygen queries from $A_2$ using the punctured key sk.
- $D'$ finally receives a bit $b'$ from $A_2$. Let the single keygen query on the target id by $A_1 \cup A_2$ be $(\text{id}^*, f)$.
  - If there are multiple queries or if $F(k, m_0) \neq F(k, m_1)$, $D'$ outputs a random bit. Else, $D'$ outputs 1 if $b = b'$, else outputs 0.

We now compare the outputs of $D'$ in the real and ideal cases. From above, it is trivial to see that for the real execution, $D'$ outputs 1 with probability $1/2 + \alpha$. On the other hand, for the ideal execution, $D'$ and thus $A_2$ get a simulated obfuscation $\tilde{\rho}$ that is independent of $b, m_0$ and $m_1$ (except for the output $F(k, m_0) = F(k, m_1)$). Thus, it outputs 1 with probability $1/2$. Now, if $\alpha$ is non-negligible, then $D'$ breaks R3PO security.

E Details Omitted from Section 7: Private Multi-Authority ABE

E.1 Proof of Security

Lemma 4 (Restated). If there exists a CP-ABE scheme, a non-interactive UC secure commitment scheme, a puncturable signature scheme, and an R3PO scheme $O_{\text{p-MA-ABE}}$ w.r.t. $(G_{\text{p-MA-ABE}}^{\text{1}}, Q_{\text{p-MA-ABE}}^{\text{1}})$, then there exists a secure p-MA-ABE scheme.

Proof: We now prove that the construction in Figure 7 is a secure p-MA-ABE scheme satisfying Definition 8

\footnote{which is also an R3PO w.r.t. $(G_{\text{p-MA-ABE}}^{\text{2}}, Q_{\text{p-MA-ABE}}^{\text{2}})$.}
– **Correctness:** Let 
\[ (st, \{req_i\}_{i \in \mathbb{N}}) \leftarrow \text{p-MA-ABE.KeyRequest}(pk, gid, \bar{x}) \]

s.t. for each \( i \in \mathbb{N} \), \( \Theta^\text{gid}_t(\bar{x}) = 1 \), \( \text{req}_i = (\text{gid}, c_i) \) and \( st = (\text{gid}, \bar{x}, \{d_i\}_{i \in \mathbb{N}}) \). For each \( t \in \mathbb{N} \), let 
\[ \rho_t = \text{p-MA-ABE.KeyGen}(t, msk_t, \Theta^\text{gid}_t, \text{req}_t, \{\text{p-MA-ABE.mpk}_i\}_{i \in \mathbb{N}}) \].

From the correctness of the R3PO obfuscation scheme, on feeding \( d_t \) (opening of \( c_t \)) to each \( \rho_t \), they output \( \text{Sig.sign}(\text{Sig}.sk_t, \text{gid}||\bar{x}) \), and on feeding these signatures, finally output \( \text{sk}_{\text{req}} \). But, this key component is an ABE key \( \text{sk}^\text{ABE}_{k,t} \) of \( \bar{x} \) for the underlying ABE scheme \( t \). From the correctness of the ABE scheme, ABE.Decrypt on input the ABE key \( \text{sk}^\text{ABE}_{k,t} \) and ciphertext ABE.ct \( \leftarrow \text{ABE.Encrypt}(\text{ABE.mpk}_t, \phi, s_t) \) outputs the secret share \( s_t \) if \( \phi(\bar{x}) = 1 \). Finally, from the correctness of additive secret sharing, we get the message \( m = s_1 + \ldots + s_N \).

– **Receiver Privacy:** Let 
\[ (st, \{req_i\}_{i \in \mathbb{N}}) \leftarrow \text{p-MA-ABE.KeyRequest}(pk, gid, \bar{x}) \]

s.t. each \( \text{req}_i = (\text{gid}, c_i) \). The global identifier \( \text{gid} \) is independent of \( \bar{x} \), and the commitment \( c_i \) reveals nothing about \( \bar{x} \) from the hiding property of the underlying commitment scheme (even against an adversary that corrupts all the authorities).

– **Security of Encryption:**

We consider \( \text{Adv} \) to be a stateful PPT machine. **Hybrid \( \mathcal{H}_{m_0} \):** corresponds to the MA-ABE security experiment (Figure 6), where the challenger encrypts message \( m_0 \).

<table>
<thead>
<tr>
<th>Hybrid ( \mathcal{H}_{m_0} ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Receive ( H ) from ( \text{Adv} ).</td>
</tr>
<tr>
<td>• ( \forall i \in H ), ( (\text{mpk}_i, \text{msk}_i) \leftarrow \text{p-MA-ABE.SetupAuth}(1^n) ). Send ( {\text{mpk}<em>i}</em>{i \in H} ) to ( \text{Adv} ).</td>
</tr>
<tr>
<td>• Receive ( {\text{mpk}<em>i}</em>{j \in \mathbb{N}\setminus H} ) from ( \text{Adv} ).</td>
</tr>
<tr>
<td>• ( \forall j \in \text{poly}(1^n) ), receive request of the form ( {t, \text{req}, \Theta^\text{gid}<em>t}</em>{i \in H} ) from ( \text{Adv} ), where ( t \in H ). Send ( \rho ), where ( \rho \leftarrow \text{p-MA-ABE.KeyGen}(t, msk_t, \Theta^\text{gid}_t, \text{req}, {\text{mpk}<em>i}</em>{i \in \mathbb{N}}) ).</td>
</tr>
<tr>
<td>• Receive challenge policy ( \phi ), messages ( (m_0, m_1) ) from ( \text{Adv} ). Send ( ct \leftarrow \text{p-MA-ABE.Encrypt}({\text{mpk}<em>i}</em>{i \in \mathbb{N}}, \phi, m_0) ) to ( \text{Adv} ).</td>
</tr>
<tr>
<td>• ( \forall j \in \text{poly}(1^n) ), receive request of the form ( {t, \text{req}, \Theta^\text{gid}<em>t}</em>{i \in H} ) from ( \text{Adv} ), where ( t \in H ). Send ( \rho ), where ( \rho \leftarrow \text{p-MA-ABE.KeyGen}(t, msk_t, \text{req}, \Theta^\text{gid}_t, {\text{mpk}<em>i}</em>{i \in \mathbb{N}}) ).</td>
</tr>
<tr>
<td>• Receive a bit ( b' ) from ( \text{Adv} ).</td>
</tr>
</tbody>
</table>

**Hybrids \( \mathcal{H}^1_k \) and \( \mathcal{H}^2_k \):** Let \( \ell \) be the total number of keygen queries by the \( \text{Adv} \). We rewrite the experiment by using an interaction between \( G^1_H \) and \( Q_k \), where we define \( Q_k \) for \( k \in [\ell] \), that receives the state \( st_{Adv} \) of the adversary \( \text{Adv} \), as follows.

**Adversary \( Q_k \):**

| • Sample commitment setups in extractable mode: for \( i \in H \), \( (st_i, \text{Com.crs}_i) \leftarrow E_0^{\text{Com}}(1^n) \). Sample public keys: \( \forall i \in H \), \( (\text{Sig}.vk_i, \text{Sig}.sk_i) \leftarrow \text{Sig.gen}(1^n) \), \( \{\text{ABE.mpk}_i, \text{ABE.msk}_i\} \leftarrow \text{ABE.Setup}(1^n) \). Send \( \{\text{mpk}_i := (\text{Com.crs}_i, \text{Sig}.vk_i, \text{ABE.mpk}_i)\}_{i \in H} \) to \( \text{Adv} \). Receive \( \{\text{mpk}_i\}_{j \in \mathbb{N}\setminus H} \) from \( \text{Adv} \). |

69
• Send \{mpk_i\}_{i \in [N]} to \mathcal{G}^j_H.

• Queries:
  * on receiving keygen query \(j < k\) of the form \((t, \text{req}, \{\Theta_t^{\text{id}}\}_{i \in H})\) from Adv, where \(t \in H, \text{req} = \text{gid}, c\).
    - extract \(\hat{x} \leftarrow \mathcal{E}^\text{com}_t(\text{st}, \text{Com.crs}_t, c)\).
    - Fix \(\pi^{(\alpha)} \in \mathcal{P}_{\text{p-MA-ABE}}\) where \(pp = (t, \text{Com.crs}_t, c), \alpha = \Theta_t^{\text{id}}\), and start = \(\sigma^1\}_{i \in [N] \setminus \text{gid}^t}.
    - Fix \(\mu^{(\beta)} \in \mathcal{M}_{\text{p-MA-ABE}}\) where \(\beta = (1, \tau, \text{ABE}.msk_t)\), where \(\tau = \text{Sig}.\text{sign}((\text{Sig}.sk_t, \hat{x})\).
    - construct \(\hat{\rho} \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi^{(\alpha)}; \mu^{(\beta)}).\) send \(\hat{\rho}\) to Adv.
  * on receiving challenge query of the form \((\hat{\phi}, m_0, m_1)\) from Adv. Send \(ct \leftarrow \text{p-MA-ABE}.\text{Encrypt} \{\{\text{mpk}_i\}_{i \in [N]}, \hat{\phi}, m_0\}\) to Adv.

• on receiving keygen query \(k\) of the form \((t, \text{req}, \{\Theta_t^{\text{id}}\}_{i \in H})\) from Adv, where \(t \in H.\) Send \((t, \text{req}, \{\Theta_t^{\text{id}}\}_{i \in H})\) as the target query and \((\text{Sig}.sk_t, \text{ABE}.msk_t)\) to \(\mathcal{G}^j_H\).

• halt with output: query \(k\), keys \{((\text{st}_t, \text{Com.crs}_t, \text{Sig}.vk_t, \text{Sig}.sk_t, \text{ABE}.mpk_t, \text{ABE}.msk_t))\}_{i \in H},\) adversary state \(st_{\text{adv}}\).

\hspace{1cm} \textbf{Hybrid} \mathcal{H}^j_{b^t}:

• Receive \(H\) from Adv.

• Run \((\mathcal{G}^j_H : Q_b)\) and get as output \((\pi^{(\alpha)}, \mu^{(\beta)}, a_G; a_Q)\).

• Parse \(a_G\) as \(\_\), parse \(a_Q\) as \((t, \text{gid}, c, \{\Theta_t^{\text{id}}\}_{i \in H})\), \((\text{st}_t, \text{Com.crs}_t, \text{Sig}.vk_t, \text{Sig}.sk_t, \text{ABE}.mpk_t, \text{ABE}.msk_t)\)}_{i \in H}, \(st_{\text{adv}}\) and load the state of Adv.

• Construct \(\rho \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi^{(\alpha)}; \mu^{(\beta)}).\) Send \(\rho\) to Adv.

• Queries:
  * on receiving keygen query \(j > k\) of the form \((t, \text{req}, \{\Theta_t^{\text{id}}\}_{i \in H})\) from Adv, where \(t \in H.\) Send \(\rho,\) where
    \[\rho \leftarrow \text{p-MA-ABE}.\text{KeyGen} \bigl(t, \text{msk}_t, \Theta_t^{\text{id}}, \text{req}, \{\text{mpk}_i\}_{i \in [N]}\bigr)\]
  * on receiving challenge query of the form \((\hat{\phi}, m_0, m_1)\) from Adv. Send \(ct \leftarrow \text{p-MA-ABE}.\text{Encrypt} \{\{\text{mpk}_i\}_{i \in [N]}, \hat{\phi}, m_0\}\) to Adv.

• Receive a bit \(b^t\) from Adv.
Hybrid $\mathcal{H}^2_k$:  
- Receive $H$ from Adv.
- Run $(G^1_H : Q_k)$ and get as output $(\pi^{(\alpha)}, \mu^{(\beta)}; a_G; a_Q)$.
- Parse $a_G$ as $\lambda$, parse $a_Q$ as $(t_1, gid, c, \{\Theta^{\text{ed}}_k\}_{i \in H^k})$, $\{\text{st}_{k}, \text{Com.crs}_{k}, \text{Sig.vk}_k, \text{Sig.sk}_k, \text{ABE.mpk}_k, \text{ABE.msk}_k\}_{k \in H^k}$, $\text{st}_{\text{Adv}}$, and load the state of Adv.
- Extract $\bar{x} \leftarrow E^\text{com}_{\text{key}}(\text{st}, \text{Com.crs}, c)$. Fix $\mu^{(\beta)'} \in M_{\mu,\text{MA-ABE}}$ with $\beta' = (1, \tau, \text{ABE.mpk}_i)$, where $\tau = \text{Sig.sign}(\text{Sig.sk}_1, \text{gid} \bar{x})$.
- Construct $\hat{\mu} \leftarrow C_{p,\text{MA-ABE}}(\pi^{(\alpha)}, \mu^{(\beta)'})$. Send $\hat{\mu}$ to Adv.
- Queries:  
  * on receiving keygen query $j > k$ of the form $(t, \text{req}, \{\Theta^{\text{ed}}_k\}_{i \in H})$ from Adv, where $t \in H$. Send $\mu$, where 
    $$\rho \leftarrow p, \text{MA-ABE.KeyGen}(t, \text{msk}_1, \Theta^{\text{ed}}_k, \text{req}, \{\text{mpk}_i\}_{i \in \mathbb{N}})$$
  * on receiving challenge query of the form $(\phi, m_0, m_1)$ from Adv. Send $ct \leftarrow p, \text{MA-ABE.Encrypt}((\{\text{mpk}_i\}_{i \in \mathbb{N}}, \phi, m_0)$ to Adv
- Receive a bit $b'$ from Adv.

Indistinguishability of $\mathcal{H}^3_m$ and $\mathcal{H}^2_k$: we have $\mathcal{H}^3_m \equiv \mathcal{H}^1_1$, and $\forall k \in \mathbb{N}$, hybrids $\mathcal{H}^2_{k-1} \equiv \mathcal{H}^1_k$, redrawing of boundaries or rephrasing. Now, we prove that $\forall k \in \mathbb{N}$, $\mathcal{H}^1_k \approx \mathcal{H}^2_k$. Invoking the R3PO security, there exists extractor $E$ and simulator $\text{Sim}$ s.t. in the experiment $(G^1_H : Q_k|E)$, $E$ outputs $(a_Q, \Pi, X^*)$ and $\rho_k \approx \text{Sim}(\pi, \mu, \Pi, \{\mu(\beta)\}(\sigma))_{\sigma \in \text{REACH}_H(X^*)}$. But, $\{\mu(\beta)\}(\sigma))_{\sigma \in \text{REACH}_H(X^*)} = \{\mu(\beta)'(\sigma))_{\sigma \in \text{REACH}_H(X^*)}$. Thus, $\rho_k \approx \hat{\rho}_k$.

Hybrids $\mathcal{H}^3_m$ and $\mathcal{H}^4_n$: We rewrite the experiment by using an interaction between $G^2_H$ and $Q'_k$, where we define $Q'_k$ as follows.

Adversary $Q'_1$:
- Sample commitment setup: $\forall i \in H$, $(\text{st}_i, \text{Com.crs}_i) \leftarrow E^\text{com}_{\text{setup}}(1^n)$, abe keys: $\forall k \in H$, $(\text{ABE.mpk}_k, \text{ABE.msk}_k) \leftarrow \text{ABE.Setup}(1^n)$. Send $\{\{\text{Com.crs}_i, \text{ABE.mpk}_i\}_{i \in H}$ to $G^2_H$. Receive $\{\text{Sig.vk}_i\}_{i \in H}$ from $G^2_H$ and send to Adv. Receive $\{\text{mpk}_i\}_{i \in \mathbb{N}\setminus H}$ from Adv and send to $G^2_H$.
- on receiving keygen query 1 of the form $(t, \text{gid}, c, \{\Theta^{\text{ed}}_k\}_{i \in H})$ from Adv, where $t \in H$. Extract $\bar{x} \leftarrow E^\text{com}_{\text{key}}(\text{st}, c)$, send $(t, \text{gid}, c, \bar{x}, \{\Theta^{\text{ed}}_k\}_{i \in H})$ as a query to $G^2_H$, receive $\tau = \{\tau_i\}_{i \in H}$ from $G^2_H$ and send $(t, \text{gid}, c, \bar{x}, \tau_i, \{\Theta^{\text{ed}}_k\}_{i \in H})$ as the target request to $G^2_H$.
- halt with output $\{\{\text{st}_i, \text{Com.crs}_i, \text{ABE.mpk}_i, \text{ABE.msk}_i, \{\text{mpk}_i\}_{i \in \mathbb{N}\setminus H}\cdot \text{st}_{\text{Adv}}\}$

Adversary $Q'_2$:
- Let $E_{k-1}$ be a reach extractor for $Q'_{k-1}$ w.r.t. generator $G^2_H$.
- Internally run $Q'_{k-1}|E_{k-1}$.
- Receive $\{\{\text{Com.crs}_i, \text{ABE.mpk}_i\}_{i \in H}$ from $Q'_{k-1}$ and forward it to $G^2_H$. Receive $\{\text{Sig.vk}_i\}_{i \in H}$ from $G^2_H$ and forward it to $Q'_{k-1}$. Receive $\{\text{mpk}_i\}_{i \in \mathbb{N}\setminus H}$ from $Q'_{k-1}$ and forward it to $G^2_H$.
- Queries:  
  * on receiving query $j < k$ of the form $(t, \text{gid}, c, \bar{x}, \{\Theta^{\text{ed}}_k\}_{i \in H})$ from $Q'_{k-1}$, forward it to $G^2_H$, receive $\tau$ from $G^2_H$ and forward it to $Q'_{k-1}$.
  * on receiving target query $j = k - 1$ of the form $(t, \text{gid}, c, \bar{x}, \tau_1, \{\Theta^{\text{ed}}_k\}_{i \in H})$ from $Q'_{k-1}$:  
    - Get $(a_Q, \Pi, X^*)$ as the output of $Q'_{k-1}|E_{k-1}$ after it halts.
· Parse \( a_Q \) as \( \{ (\text{st}_i, \text{Com.crs}_i, \text{ABE.mpk}_i, \text{ABE.msk}_i, \{ \text{mpk}_i \} ) \}_{i \in [N] \setminus H}, st_{adv} \) and load the state of Adv.
· Construct \( \tilde{\rho} \leftarrow \text{Sim}_{p-MA-ABE}(\pi, \mu, \Pi, \{ \mu(\delta) \} )_{\sigma \in \text{REACH}_H(X^*)} \). Send \( \tilde{\rho} \) to Adv.
  * on receiving challenge query of the form \((\phi, m_0, m_1)\) from Adv. Send \( ct \leftarrow \text{p-MA-ABE.Encrypt} \{ \{ \text{mpk}_i \} )_{i \in [N], \phi, m_0} \) to Adv
  * on receiving query \( j = k \) of the form \((t, \text{gid}_c, \{ \text{Theta}^d \} )_{i \in H} \) from Adv.
    · Extract \( \hat{x} \leftarrow E^E_{\text{Com}}(\text{st}_i, c) \) and send \((t, \text{gid}_c, \hat{x}, \{ \text{Theta}^d \} )_{i \in H} \) as a request to \( G^2_H \). Receive \( \tau = \{ \tau_i \}_{i \in H} \) from \( G^2_H \). Send \((t, \text{gid}_c, \hat{x}, \tau_i, \{ \text{Theta}^d \} )_{i \in H} \) as the target request to \( G^2_{p-MA-ABE} \).
    * halt with output \((\{ (\text{st}_i, \text{Com.crs}_i, \text{ABE.mpk}_i, \text{ABE.msk}_i, \{ \text{mpk}_i \} ) )_{i \in [N] \setminus H}, st_{adv} \).

### Hybrid \( H^3_H \):

- Receive \( H \) from Adv.
- Run \((G^2_H : Q'_1)\) and get as output \((\pi^{(\alpha)}, \mu^{(\beta)}, a_Q; a_Q)\).
- Parse \( a_Q \) as \((\{ \text{Sig.vk}_i, \text{Sig.sk}_i \} )_{i \in H}, \text{parse} a_Q \) as \((\{ (\text{st}_i, \text{Com.crs}_i, \text{ABE.mpk}_i, \text{ABE.msk}_i, \{ \text{mpk}_i \} ) )_{i \in [N] \setminus H}, st_{adv} \) and load the state of Adv.
- Construct \( \rho \leftarrow O_{p-MA-ABE}(\pi^{(\alpha)}, \mu^{(\beta)}) \). Send \( \rho \) to Adv.
- Queries:
  * on receiving keygen query \( j > k \) of the form \((t, \text{req}, \{ \text{Theta}^d \} )_{i \in H} \) from Adv, where \( t_j \in H \) and \( \text{req} = (\text{gid}_c) \):
    · extract \( \hat{x} \leftarrow E^E_{\text{Com}}(\text{st}_i, \text{Com.crs}_i, c) \).
    · Fix \( \pi^{(\alpha)} \in P_{p-MA-ABE} \) where \( pp = (t, \text{Com.crs}_i, c), \alpha = \Theta_i^{(\delta)}, \) and start = \( \sigma_i^{(\text{mpk}_i)} \) \( i \in [N] \setminus \text{gid} \).
    · Fix \( \mu^{(\beta)} \in M_{p-MA-ABE} \) where \( \beta = (1, \tau_i, \text{ABE.msk}_i) \), where \( \tau_i = \text{Sig.sign}(\text{Sig.sk}_i, \hat{x}) \).
    · construct \( \hat{\rho} \leftarrow O_{p-MA-ABE}(\pi^{(\alpha)}, \mu^{(\beta)}) \) send \( \hat{\rho} \) to Adv.
  * on receiving challenge query of the form \((\phi, m_0, m_1)\) from Adv. Send \( ct \leftarrow \text{p-MA-ABE.Encrypt} \{ \{ \text{mpk}_i \} )_{i \in [N], \phi, m_0} \) to Adv
  * Receive a bit \( b' \) from Adv.

### Hybrid \( H^3_H \):

- Receive \( H \) from Adv.
- Run \((G^2_H : Q'_1)\) and get as output \((\pi^{(\alpha)}, \mu^{(\beta)}, a_Q; a_Q, \Pi, X^*)\).
- Parse \( a_Q \) as \((\{ \text{Sig.vk}_i, \text{Sig.sk}_i \} )_{i \in H}, \text{parse} a_Q \) as \((\{ (\text{st}_i, \text{Com.crs}_i, \text{ABE.mpk}_i, \text{ABE.msk}_i, \{ \text{mpk}_i \} ) )_{i \in [N] \setminus H}, st_{adv} \) and load the state of Adv.
- Construct \( \tilde{\rho} \leftarrow \text{Sim}_{p-MA-ABE}(\pi, \mu, \{ \mu^{(\delta)}(\sigma) \} )_{\sigma \in \text{REACH}_H(X^*)} \). Send \( \tilde{\rho} \) to Adv.
- Queries:
  * on receiving keygen query \( j > k \) of the form \((t, \text{req}, \{ \text{Theta}^d \} )_{i \in H} \) from Adv, where \( t \in H \) and \( \text{req} = (\text{gid}_c) \):
    · extract \( \hat{x} \leftarrow E^E_{\text{Com}}(\text{st}_i, \text{Com.crs}_i, c) \).
    · Fix \( \pi^{(\alpha)} \in P_{p-MA-ABE} \) where \( pp = (t, \text{Com.crs}_i, c), \alpha = \Theta_i^{(\delta)}, \) and start = \( \sigma_i^{(\text{mpk}_i)} \) \( i \in [N] \setminus \text{gid} \).
    · Fix \( \mu^{(\beta)} \in M_{p-MA-ABE} \) where \( \beta = (1, \tau_i, \text{ABE.msk}_i) \), where \( \tau_i = \text{Sig.sign}(\text{Sig.sk}_i, \hat{x}) \).
    · construct \( \hat{\rho} \leftarrow O_{p-MA-ABE}(\pi^{(\alpha)}, \mu^{(\beta)}) \) send \( \hat{\rho} \) to Adv.
  * on receiving challenge query of the form \((\phi, m_0, m_1)\) from Adv. Send \( ct \leftarrow \text{p-MA-ABE.Encrypt} \{ \{ \text{mpk}_i \} )_{i \in [N], \phi, m_0} \) to Adv
  * Receive a bit \( b' \) from Adv.
Lemma 17. \((\mathcal{G}_H^1 : \mathcal{Q})\) decomposes into

\[
\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3.0, \ldots, \mathcal{L}_{3.0}),
\]

where:

\[
\mathcal{L}_i = \begin{cases} 
(\mathcal{G}_{\text{COMMIT}}^\mathcal{Q}) & \text{if } i = 1 \\
(\mathcal{G}_\Sigma^\mathcal{Q}) & \text{if } i = 2 \\
(\mathcal{G}_{\text{SIGN}}^\mathcal{Q}) & \text{if } i = (3, a), a + 1 \in [N]
\end{cases}
\]

Proof: We prove that \(\forall G_H^1 \in \mathcal{G}_H^1 : \mathcal{Q}, G_H^1\) decomposes to \(\mathcal{L}\).

Case \(i = 1\):

At this partition, we prove that \((G_H^1, \mathcal{Q})\) decomposes to \((\mathcal{G}_{\text{COMMIT}}^\mathcal{Q})\) (with the appropriate prefix and partition function, that we skip here; refer Appendix C.5):

- **Defining J**: It extracts \(pk, \text{gid}, pp\) from the transcript of the interaction and halts with output \((\hat{\pi}_{\sigma_1}^{(a)}, \hat{\mu}^{(b)})\).

- **Defining Z**: It interacts with \(Q\) by internally running \(G_H^1\). Finally, it outputs \(G_H^1\)'s output.

- **Defining W**: It extracts \(c\) from the transcript of the interaction and sends \(c\) on the bottom channel.

Case \(i = 2\): At this partition, we prove that \((G_H^1, \mathcal{Q})\) decomposes to \((\mathcal{G}_\Sigma^\mathcal{Q})\) (with the appropriate prefix and partition function, that we skip here; refer Appendix C.5):

- **Defining J**: It receives \(\sigma_1, \sigma_2\) from \(W\), samples a message function \(\hat{\mu}^{(b)} \in \hat{M}_{p-\text{MA-ABE}, 1}\) and halts with output \((\hat{\pi}_{\sigma_2}^{(a)}[\sigma_1], \hat{\mu}^{(b)})\).

73
We prove that

Proof:

At this partition, we prove that \( G^1_H \) decomposes to \((G^\text{Sign}, Q)\) (with the appropriate prefix and partition function, that we skip here; refer Appendix C.5):

- **Defining** \( J \): It extracts \( \sigma = \sigma'^{i}_{pk, gid, \bar{x}} \) from the transcript of the interaction, samples a message function \( \hat{\mu}(\bar{\beta}) \) and halts with output \((\pi^{(\alpha)}_{\sigma}, \hat{\mu}(\bar{\beta}))\).

- **Defining** \( Z \): It interacts with \( Q \) by internally running \( G^2_H \). Finally, it outputs \( G^1_H \)'s output.

- **Defining** \( W \): It extracts \( \text{Sig, }vk_{i+1}, gid, \bar{x} \) from the transcript of the interaction and sends \( \text{Sig, }vk_{i+1}, gid||\bar{x} \) on the bottom channel.

\[ \square \]

**Lemma 18.** \((G^2_{p-MA-ABE}, Q_{p-MA-ABE})\) decomposes into \( \mathcal{L} = \{ \mathcal{L}_H \}_H \), where, for each \( H \) (set of honest authorities),

\[ \mathcal{L}_H = (\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_{3.0}, \ldots, \mathcal{L}_{3.N}) \],

and \( \forall i \in [3+N] : \)

\[ \mathcal{L}_i = \begin{cases} G^\text{Com}_i(Q) & \text{if } i = 1 \\ G^\Sigma_i(Q) & \text{if } i = 2 \\ G^\text{Sign}_i(Q) & \text{if } i = (3,a), a + 1 \in [N] \setminus H \\ G^\text{Sign}_i(Q) & \text{if } i = (3,a), a + 1 \in H \end{cases} \]

**Proof:** We prove that \( \forall G^2_H \in G^2_{p-MA-ABE}, G^2_H \) decomposes to \( \mathcal{L}_H \). The first three cases are similar to Lemma 17.

**Case** \( i = (3,a), a + 1 \in H : \)

At this partition, we prove that \((G^2_H, Q)\) decomposes to \((G^\text{Sign}, Q)\) (with the appropriate prefix and partition function, that we skip here; refer Appendix C.5):

- **Defining** \( W \): It internally runs \( Q \) in a white-box way. For each query \( m^j \) that \( Q \) makes in the top channel, it sends on the bottom channel and gets back \( \tau_j \). Finally, \( Q \) sends the target message \( m \) on the top channel. It extracts \( gid, \bar{x} \) from \( m \) and sends as the target message on the bottom channel.

- **Defining** \( J \): It responds to each query from \( W \) using the corresponding response of \( G^2_H \). It finally receives \( gid, \bar{x} \) from \( W \) as the target message, samples a message function \( \hat{\mu}(\bar{\beta}) \) and outputs \((\pi^{(\alpha)}_{\sigma^3_{a, \bar{x}}}, \hat{\mu}(\bar{\beta}))\).

- **Defining** \( Z \): It samples the signature keys for all honest authorities except \( i + 1 \). It responds to each query of \( Q \) using these keys. If \( Q \) queries on the key for \( i + 1 \), \( W \) makes a corresponding query in the bottom channel and gets the corresponding response. \( Z \) uses this to respond to \( Q \). Finally, \( Z \) halts with output \((\pi^{(\alpha)}, \mu^{(\beta)})\) (with the appropriate parameters).

\[ \square \]

We now prove the main lemma.
Lemma 5 (Restated). If there exist R3PO schemes w.r.t. $(\mathcal{G}_{\text{Com}}^{\text{Com}}, \mathcal{Q}_{\text{Com}}^{\text{Com}}, \mathcal{P}_{\text{Com}}^{\text{Com}})$ and $(\mathcal{G}_{\text{Sig}}^{\text{Sig}}, \mathcal{Q}_{\text{Sig}}^{\text{Sig}}, \mathcal{P}_{\text{Sig}}^{\text{Sig}})$, where Com is a non-interactive UC-secure commitment scheme and Sig is a puncturable signature scheme, then there exists a compiler $\mathcal{O}_{\text{p-MA-ABE}}$ which is an R3PO scheme w.r.t. $(\mathcal{G}_{\text{p-MA-ABE}}^{1}, \mathcal{Q}_{\text{p-MA-ABE}}^{1})$, and also an R3PO w.r.t. $(\mathcal{G}_{\text{p-MA-ABE}}^{2}, \mathcal{Q}_{\text{p-MA-ABE}}^{2})$.

Proof: If there exists a semi-honest secure OTSE scheme, then there exists a PPT program $\mathcal{O}_{\text{Sig}}$ s.t. $\mathcal{O}_{\text{Sig}}$ is a R3PO scheme w.r.t. $(\mathcal{G}_{\text{Sig}}^{\text{Sig}}, \mathcal{Q}_{\text{Sig}}^{\text{Sig}}, \mathcal{P}_{\text{Sig}}^{\text{Sig}})$ (Lemma 2) and $\mathcal{O}_{\text{Sig}}$ is a R3PO scheme w.r.t. $(\mathcal{G}_{\text{QSIG}}^{\text{QSIG}}, \mathcal{Q}_{\text{QSIG}}^{\text{QSIG}}, \mathcal{P}_{\text{QSIG}}^{\text{QSIG}})$ (Lemma 10).

Now, from our compiler (Theorem 1), Lemma 17 and Lemma 18 and Lemma 10, there exists a PPT program $\mathcal{O}_{\text{p-MA-ABE}}$ s.t. $\mathcal{O}_{\text{p-MA-ABE}}$ is a R3PO obfuscation scheme w.r.t. $(\mathcal{G}_{\text{p-MA-ABE}}^{1}, \mathcal{Q}_{\text{p-MA-ABE}}^{1})$ and $(\mathcal{G}_{\text{p-MA-ABE}}^{2}, \mathcal{Q}_{\text{p-MA-ABE}}^{2})$. $\square$

F MPC from R3PO for Commitment-Opening

In this section, we give our 2-round MPC construction, which is a simpler redereivation of the results in [7, 27] using an R3PO scheme for commitment opening (see Section 4.1) and our composition theorem.

F.1 Conforming Protocols

An MPC protocol for a functionality $\mathcal{F}$ is a protocol where a group of $n$ mutually distrusting parties $P_1, \ldots, P_n$, with private inputs $x_1, \ldots, x_n$ respectively, interact with each other to achieve this functionality. Conforming protocols are special kind of MPC protocols which satisfy the following properties:

- The topology of a circuit computing $\mathcal{F}$, with $L = \sum_{i} |x_i|$ input gates (indexed as $1, \ldots, L$), $T$ internal binary gates (indexed as $L+1, \ldots, L+T$) and $K$ binary output gates, is specified by a function $\phi$, as follows: for each binary gate index $w$, $\phi_w = (i, u, v)$ denotes that the gate’s inputs are the outputs of gates $u$ and $v$ (for some $u, v < w$); further, this gate is “owned” by party $P_i$. Let $A_i$ denote the set of binary gate indices owned by $P_i$; i.e., $A_i = \{w \mid \phi_w = (i, \cdot, \cdot)\}$.

- The protocol $\Phi$ is divided into 3 phases.
  - In the first phase, every party $P_i$, based on its input $x_i$ and private randomness, generates private gates $\{G_w\}_{w \in A_i}$ and a string $z_i$ such that $|z_i| = |x_i|$. It then broadcasts $z_i$. A common state vector $s$ is initialized as $z_1 || \cdots || z_n$ (at this point $s$ will be $L$ bits long).
  - The second phase consists of $T$ rounds. In round $w \leq L$, if $\phi_w = (i, u, v)$, party $P_i$ broadcasts the bit $s_w = G_w(s_u, s_v)$, which is appended to $s$.
  - In the third phase, for each output gate $w$ such that $\phi_w = (i, u, v)$, $P_i$ computes its value as $G_w(s_u, s_v)$ and outputs it.

Just like is done in [27], it can be shown that any MPC protocol can be converted to a conforming protocol with at most a polynomial blowup in the round complexity of the protocol. If the underlying MPC protocol is secure against active corruption, then the resulting protocol remains secure against active corruption in Phase 1 (so that $z_i$ and $\{G_w\}_{w \in A_i}$ maybe arbitrary for corrupt $P_i$) and passive corruption in Phase 2.

F.2 MPC Construction

We will build a 2-round MPC protocol assuming a multi-round conforming protocol $\Phi$ for the same functionality, a commitment scheme, and an R3PO scheme $\mathcal{O}_{\text{Com}}$ w.r.t. $(\mathcal{G}_{\text{Com}}^{\text{Com}}, \mathcal{Q}_{\text{Com}}^{\text{Com}}, \mathcal{P}_{\text{Com}}^{\text{Com}})$. We only need a commitment scheme where the messages to be committed are bits i.e., $M = \{0,1\}$. First, we must

\[^{25}\text{Defined in Section 4.1.}\]
define the program classes, the generator class and the adversary class w.r.t. which we will define our transition programs and their obfuscations. The party index $i$ is fixed for each class, as each party’s programs have similar transitions in different orders.

Here is a detailed description of the classes:

- Program family $\mathcal{P}^{\text{Com},i}_{\text{MPC}}$: Each program in the program family is parameterized by a conforming protocol $\Phi$, a vector of common random strings $\text{crs} = \{\text{crs}_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$, a vector of commitments $c = \{c_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$ and an index $i$. The state space is defined as

$$\Sigma_{\text{MPC}} = \{\sigma_{s}^{w-L}\}_{w \in \{L,...,L+T+1\}, s \in \{0,1\}^w}$$

A program $\pi_{\Phi,\text{crs},c,i}[\sigma_{s}^{0}] \in \mathcal{P}^{\text{Com},i}_{\text{MPC}}$, where $\alpha = \{G_{w}\}_{w \in A_{i}}$, is defined by the transition function:

$$\pi_{\Phi,\text{crs},c,i}[\sigma_{s}^{0}] = \begin{cases} \sigma_{s}^{w-L} & \text{if } \phi_{w} = (i, u, v), \\ \sigma_{s'}^{w-L} & \text{if } \phi_{w} = (i^*, u, v), \\ \sigma_{s}^{w-L-1} & \text{otherwise.} \end{cases}$$

for $w \in \{L : T\}$. We shall also associate a partition function $I_{\text{MPC}} : \Sigma \rightarrow \{0, \ldots, T + 1\}$, such that $I_{\text{MPC}}(\sigma_{s}^{w-L}) = w - L$, for $w \in \{L, \ldots, L + T + 1\}$.

![Diagram](image)

Fig. 28. Top figure shows the transition program $\pi_{\Phi,\text{crs},c,i}[\sigma_{s}^{0}]$ produced by $P_{i}$, when $L+2$, $L+5 \in A_{i}$ and $L+1, L+3, L+4 \not\in A_{i}$. The details of the transitions for $w$ such that $\phi_{w} = (j, u, v)$, where $j \neq i$ (middle) and $j = i$ (bottom) are also shown.
- Program family $\hat{P}_{\text{MPC}}$: This program family is essentially the same as $P_{\text{MPC}}^{\text{Com},i}$ where Com is a trivial, non-hiding commitment scheme (i.e., Com.Commit$(m) = m$). In other words, a program $\Pi_{\Phi,x}[\sigma^0_{s0}] \in \hat{P}_{\text{MPC}}$ is defined by the transition function:

$$\Pi_{\Phi,x}[\sigma^0_{s0}](\sigma^w_{s-L^{-1}}, x) = \begin{cases} 
\sigma^w_{s'-L} & \text{if } \phi_w = (\cdot, u, v) \text{ and } x = x_t, \\
\sigma^w_{s-L} & \text{where } s' = s||x. 
\end{cases}$$

where $x \in \{0, 1\}^T$.

![Diagram](image.png)

Fig. 29. The top figure denotes the relaxed transition program $\Pi_{\Phi,x}[\sigma^0_{s0}]$, with a particular transition for $w$ s.t. $\phi_w = (\cdot, u, v)$ being shown in the bottom figure.

- A $(P_{\text{MPC}}^{\text{Com},i}, \hat{M}_{\text{MPC},1})$-Generator Class $G_{\text{MPC}}^{\text{Com},i}$. This class contains all generators of the form $H_{\text{MPC}}^{\text{Com},i}$$Z$, for all PPT $Z$, where $H_{\text{MPC}}^{\text{Com},i}$ behaves as follows: It receives $\text{crs} = \{\text{crs}_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$ from $Q$. It then generates $(z_i, \{G_w\}_{w \in A_i}) \leftarrow \text{pre}(1^k, i, x_i)$, with $x_i$ being an input to $H_{\text{MPC}}^{\text{Com},i}$. For every $w \in A_i$, and $\gamma, \delta \in \{0,1\}$, $H_{\text{MPC}}^{\text{Com},i}$ generates

$$(c_w, \gamma, \delta, d_w, \gamma, \delta) \leftarrow \text{Com.Commit}(\text{crs}_{w,\gamma,\delta}, G_w(\gamma, \delta)).$$

When interacting with $Q$, it sends $(z_i, \{c_w,\gamma,\delta\}_{w \in A_i, \gamma, \delta \in \{0,1\}})$ to $Q$. It then reads $(z_j, \{c_w,\gamma,\delta\}_{w \in A_j, \gamma, \delta \in \{0,1\}})_{j \in [n]}\setminus\{i\}$ from $Q$. Also, $H_{\text{MPC}}^{\text{Com},i}$ accepts a message function $\hat{\mu}(\cdot) \in \hat{M}_{\text{MPC},1}$ from $Z$. Let $c = \{c_w,\gamma,\delta\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$. $H_{\text{MPC}}^{\text{Com},i}$ outputs $(\pi_{\Phi,c,c}^{(i)}[\sigma^0_{s0}], \hat{\mu}(\cdot))$, where $s^0 = z_1||\ldots||z_N$. $(Z$’s output forms the auxiliary output of the generator $H_{\text{MPC}}^{\text{Com},i}$$Z$.)

- Adversary Class $Q_{\text{MPC}}^{\text{Com},i}$: It is the class of all PPT adversaries who must send the common random string $\text{crs} = \{\text{crs}_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$ and messages of the form $(z_j, \{c_w,\gamma,\delta\}_{w \in A_j, \gamma, \delta \in \{0,1\}})_{j \in [n]}\setminus\{i\}$ to $H_{\text{MPC}}^{\text{Com},i}$.  

77
Next, we will show that the above classes decompose\(^{26}\) to the corresponding classes for commitment opening and epsilon-transitions in the library, which already have R3PO schemes\(^{27}\). It is useful to do this so that we can use our main theorem Theorem 1 to obtain a valid R3PO scheme for these MPC classes as a corollary.

**Lemma 19.** \((G_{\text{MPC}}^{\text{Com},i}, Q_{\text{MPC}}^{\text{Com},i})\), for every \(i \in [n]\), decomposes into a library of commitment openings \((G_{\text{COMMIT}}^{\text{Com}}, Q_{\text{COMMIT}}^{\text{Com}})\) and epsilon-transitions \((G_{\epsilon}^{\Sigma}, Q_{\epsilon}^{\Sigma})\) (where, \(\Sigma = \Sigma_{\text{MPC}}\)).

**Proof:** We will show that \((G_{\text{MPC}}^{\text{Com},i}, Q_{\text{MPC}}^{\text{Com},i})\) decomposes into the library \(L = (L_{w-L})_{w \in \{L:T\}}\), where

\[
L_{w-L} = \begin{cases} 
(G_{\text{COMMIT}}^{\text{Com}}, Q_{\text{COMMIT}}^{\text{Com}}) & \text{if } w \notin A_i, \\
(G_{\epsilon}^{\Sigma}, Q_{\epsilon}^{\Sigma}) & \text{otherwise},
\end{cases}
\]

where \(\Sigma = \Sigma_{\text{MPC}}\). First, we will show the decomposition at partitions \(w \notin A_i\). Therefore, for a generator \(G \in G_{\text{MPC}}^{\text{Com},i}\) and a partition \(w \notin A_i\), we will describe \(J, Z\) so that \(\forall Q \in G_{\text{MPC}}^{\text{Com},i}\), and all \((w-L)\)-partial reach-extractors \(\mathcal{E}\) for \(Q\), the conditions given in Definition 5 hold.

---

**Defining J.** Here is a description of the interaction between \(H_{\text{MPC}}^{\text{Com},i}\), \(J, Q\) and \(\mathcal{E}\):

1. \(J\) reads \(\text{crs} = \{\text{crs}_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}\) from \(Q\) and forwards it to \(H_{\text{MPC}}^{\text{Com},i}\).
2. \(J\) receives \((z_i, \{c_{w,\gamma,\delta}\}_{w \in A_i, \gamma, \delta \in \{0,1\}})\) from \(H_{\text{MPC}}^{\text{Com},i}\) and forwards it to \(Q\).
3. \(J\) receives \((\{z_j, \{c_{w,\gamma,\delta}\}_{w \in A_i, \gamma, \delta \in \{0,1\}}\})_{j \in [n]\setminus\{i\}}\) from \(Q\) and forwards it to \(H_{\text{MPC}}^{\text{Com},i}\).
4. \(J\) receives \((a_Q, \Pi_{\Phi}(\sigma^0), X^*)\) from extractor \(\mathcal{E}\). It parses \(X^* = (x_t)_{t \in [T]}\), where \(x_t\) is a bit for all \(t \in [T]\). Set \(s^0 = z_1 \parallel \ldots \parallel z_n\). For every \(t \in [T]\), \(J\) does the following:

\[
s^t = s^{t-1} \parallel x_t.
\]

where \(\phi_t + L = (\gamma, u, v)\).

5. Set \(\gamma = s^w_{w-L} - 1\) and \(\delta = s^w_{w-L} - 1\), where \(\phi_w = (i^* \neq i, u, v)\).

6. \(J\) samples a message function \(\hat{\mu}(\tilde{\beta}) \in \hat{\mathcal{M}}_{\text{MPC},w}\) and sends it to \(H_{\text{MPC}}^{\text{Com},i}\).

7. \(J\) outputs \((\tilde{\pi}_\sigma(\alpha), \hat{\pi}_\sigma(\tilde{\beta}))\) as its auxiliary output, where \(\sigma = \sigma_{w-L-1}^w\) and \(\hat{\pi}_\sigma(\alpha)\) is a one-step restriction (defined in Section 5) of the program \(\pi_{\Phi,\text{crs},\epsilon,i}\) (defined in Appendix F.2). Note that this one-step program only needs to have \(\text{crs}_{w,\gamma,\delta}\) and \(c_{w,\gamma,\delta}\) hard-coded in it.

---

**Defining Z.** Here is a description of the interaction between \(H_{\text{COMMIT}}^{\text{Com}}, Z, Q\) and \(\mathcal{E}\):

1. \(Z\) reads \(\text{crs} = \{\text{crs}_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}\) from \(Q\).
2. \(Z\) samples \((z_i, \{G_w\}_{w \in A_i})\) \(\leftarrow\) \(\text{pre}(1^*, i, x_i)\), where \(x_i\) is its input.
3. For every \(w \in A_i\), and \(\gamma, \delta \in \{0,1\}\), \(Z\) generates

\[
(c_{w,\gamma,\delta}, d_{w,\gamma,\delta}) \leftarrow \text{Com.Commit}(\text{crs}_{w,\gamma,\delta}, G_w(\gamma, \delta)).
\]

4. \(Z\) sends \((z_i, \{c_{w,\gamma,\delta}\}_{w \in A_i, \gamma, \delta \in \{0,1\}})\) to \(Q\).

5. \(Z\) receives \((\{z_j, \{c_{w,\gamma,\delta}\}_{w \in A_j, \gamma, \delta \in \{0,1\}}\})_{j \in [n]\setminus\{i\}}\) from \(Q\).

\(^{26}\) Refer to Definition 5.

\(^{27}\) These are also provided in the library.
6. \( Z \) receives \((a_Q, \Pi_{\Phi_X}(s^0), X^*)\) from extractor \( \mathcal{E} \). It parses \( X^* = (x_t)_{t \in [T]} \), where \( x_t \) is a bit for all \( t \in [T] \). Set \( s^0 = z_1 \ldots z_n \). For every \( t \in [T] \), \( Z \) does the following:

\[
s' = s^{t-1} \| x_t.
\]

where \( \phi_t + L = (u, v, \nu) \).

7. Set \( \gamma = s^{w-L-1}_w \) and \( \delta = s^{w-L-1}_w \), where \( \phi_w = (i^* \neq i, u, v) \).

8. It then provides \((c_w, \gamma, \delta, \text{crs}_{w, \gamma, \delta})\) to \( H_{\text{COMMIT}}^{\text{Com}} \).

9. \( Z \) then samples \( \tilde{\mu}^{(\beta)}(\bar{\alpha}) \in \tilde{\mathcal{M}}_{\text{COMMIT}, w} \) and sends \( \tilde{\mu}^{(\beta)}(\bar{\alpha}) \) to \( H_{\text{COMMIT}}^{\text{Com}} \).

10. \( Z \) halts with output \((\pi^{(\alpha)}_{\Phi_{\text{crs}}, c, i}, s^0_w, \tilde{\mu}^{(\beta)}(\bar{\alpha}))\), where \( s^0 = z_1 \ldots z_N \) and \( c = \{c_w, \gamma, \delta\}_{w \in \{L, T\}, \gamma, \delta \in \{0, 1\}} \).

- **Indistinguishability check.** We need to show that the outputs of \( \langle H_{\text{MPC}}^{\text{Com}, i} \| J : Q \rangle \) and \( \langle H_{\text{COMMIT}}^{\text{Com}} \| Z : Q \rangle \) are indistinguishable. This follows in a straightforward manner from the descriptions of \( J \) and \( Z \) provided above.

- **Reach-restriction check.** Since \( J \) (as defined above) is only using the input sequence \( X^* \) obtained from \( \mathcal{E} \) to determine the start state \( \sigma \) of the non-reactive program \( \hat{\pi}_{\sigma}^{(\alpha)} \), the reach-restriction condition is satisfied.

Now, we will prove decomposition for \( w \in A_i \). Consider the following description:

- **Defining \( J \).** In an interaction with \( H_{\text{MPC}}^{\text{Com}, i} \), \( Q \) and \( \mathcal{E} \), steps 1-4 are the same as the description of \( J \) for the previous case. Here are the following steps:

1. \( J \) sets \( \sigma_1 := \sigma^{w-L-1}_w \) and \( \sigma_2 := \sigma^{w-L}_w \), where the \((w-L)\)-th transition is done similarly to the first \( w-L-1 \) transitions i.e., using the output of \( \mathcal{E} \).

2. \( J \) samples a message function \( \tilde{\mu}^{(\beta)}(\bar{\alpha}) \in \tilde{\mathcal{M}}_{\text{MPC}, w} \) and sends it to \( H_{\text{MPC}}^{\text{Com}, i} \).

3. It outputs \((\pi^{(\alpha)}_{\Phi_{\text{crs}}, c, i}, s^0_w, \tilde{\mu}^{(\beta)}(\bar{\alpha}))\) as its auxiliary output, where \( \pi_{\Phi_{\text{crs}}, c, i}^{(\alpha)} \) is a one-step restriction of the program \( \pi_{\Phi_{\text{crs}}, c, i}^{(\alpha)} \) at state \( \sigma_1 \). Note that this one-step program only needs to have \( \sigma_1 \) and \( \sigma_2 \) hard-coded in it, since it is an \( \epsilon \)-transition.

- **Defining \( Z \).** In an interaction with \( H_{\text{E}}^{\text{E}}, Q \) and \( \mathcal{E} \) (where \( \Sigma = \Sigma_{\text{MPC}} \)), steps 1-6 are the same as in the description of \( Z \) for the previous case. Here are the following steps:

1. \( Z \) sets \( \sigma_1 := \sigma^{w-L-1}_w \) and \( \sigma_2 := \sigma^{w-L}_w \), where the \((w-L)\)-th transition is done similarly to the first \( w-L-1 \) transitions i.e., using the output of \( \mathcal{E} \).

2. \( Z \) samples \( \tilde{\mu}^{(\beta)}(\bar{\alpha}) \in \tilde{\mathcal{M}}_{\text{MPC}, w} \) and sends \((\sigma_1, \sigma_2, \tilde{\mu}^{(\beta)}(\bar{\alpha}))\) to \( H_{\text{E}}^{\text{E}} \).

3. \( Z \) halts with output \((\pi^{(\alpha)}_{\Phi_{\text{crs}}, c, i}, s^0_w, \tilde{\mu}^{(\beta)}(\bar{\alpha}))\), where \( s^0 = z_1 \ldots z_N \) and \( c = \{c_w, \gamma, \delta\}_{w \in \{L, T\}, \gamma, \delta \in \{0, 1\}} \).

- **Indistinguishability check.** We need to show that the outputs of \( \langle H_{\text{MPC}}^{\text{Com}, i} \| J : Q \rangle \) and \( \langle H_{\text{E}}^{\text{E}} \| Z : Q \rangle \) are indistinguishable, where \( \Sigma = \Sigma_{\text{MPC}} \). Again, this follows in a straightforward manner from the descriptions of \( J \) and \( Z \) given above.

- **Reach-restriction check.** Since \( J \) (as defined above) is only using the input sequence \( X^* \) obtained from \( \mathcal{E} \) to determine the start state \( \sigma \) of the non-reactive program \( \hat{\pi}_{\sigma}^{(\alpha)} \), the reach-restriction condition is satisfied.

\[ \blacksquare \]

**Corollary 3.** Assuming a UC-secure commitment opening scheme \( \text{Com} \), and an R3PO scheme w.r.t. \((G_{\text{COMMIT}}^{\text{Com}}, Q_{\text{COMMIT}}^{\text{Com}}, \hat{\mathcal{P}}_{\text{COMMIT}}^{\text{Com}})\), there exists an R3PO scheme w.r.t. \((G_{\text{MPC}}^{\text{Com}}, Q_{\text{MPC}}^{\text{Com}}, \hat{\mathcal{P}}_{\text{MPC}}^{\text{Com}})\), for every \( i \in [n] \).

Now, we give our construction of a 2-round MPC protocol in Figure 30. The construction is very similar to [27] except that the second round messages, consisting of multiple OT messages and a sequence of
Lemma 20. Assuming a conforming protocol $\Phi$ for a functionality, a UC-secure commitment scheme $\text{Com}$, and an $R3PO$ scheme w.r.t. $(G_{\text{MPC}}^{\text{Com},i}, Q_{\text{MPC}}^{\text{Com},i}, \hat{P}_{\text{MPC}})$ for every $i \in [n]$, there exists a secure 2-round MPC protocol for the same functionality.

Proof: Let $A$ be a malicious adversary corrupting a subset of parties and let $H \subseteq [n]$ be the set of honest parties. This set is fixed before the execution of the protocol.

**Description of the Simulator.** We give the description of $\text{Sim}$ that simulates the view of $A$. $\text{Sim}$ will internally use the simulator for malicious security of the underlying conforming protocol $\text{Sim}_\Phi$, the extractors $(E_1, E_2)$ implied by binding property of $\text{Com}$, the simulator $\text{Sim}_{E,q}$ implied by equivocal commitment property, the extractor $E_{\text{MPC}}^{\text{Com},i}$ and the simulator $\text{Sim}_{\text{MPC}}^{\text{Com},i}$ for security of $C_{\text{MPC}}^{\text{Com},i}$, for every $i \in H$.

**Simulating the interaction with $Z$.** For every input value for the set of corrupted parties that $\text{Sim}$ receives from $Z$, $\text{Sim}$ writes that value to $A$’s input tape. Similarly, the output of $A$ is written as the output on $\text{Sim}$’s output tape.

**Simulating the interaction with $A$:** $\text{Sim}$ does the following:

- **Generation of the common random string:** It generates the common random string as follows:
  - For each $i \in H$, $w \in A_i$, $\gamma, \delta \in \{0, 1\}$, set
    \[
    (\text{crs}_{w,\gamma,\delta}, c_{w,\gamma,\delta}, \{d^0_{w,\gamma,\delta}, d^1_{w,\gamma,\delta}\}) \leftarrow \text{Sim}_q(1^\kappa).
    \]
  - For each $i \in [n] \setminus H$, $w \in A_i$, $\gamma, \delta \in \{0, 1\}$, generate
    \[
    (\text{crs}_{w,\gamma,\delta}, \text{state}_{w,\gamma,\delta}) \leftarrow E_1(1^\kappa).
    \]

- **Initialization:** $\text{Sim}$ executes $\text{Sim}_\Phi(1^\kappa)$ to obtain \{\(z_i\)\}_{i \in H}.

- **Round-1 messages from $\text{Sim}$ to $A$:** For each $i \in H$, the simulator $\text{Sim}$ sends \(\{z_i, \{c_{w,\gamma,\delta}\}_{w \in A_i, \gamma, \delta \in \{0, 1\}}\}\) to $A$ on behalf of honest party $P_i$.

- **Round-1 messages from $A$ to $\text{Sim}$:** For each $i \in [n] \setminus H$, $\text{Sim}$ receives the value \(\{z_i, \{c_{w,\gamma,\delta}\}_{w \in A_i, \gamma, \delta \in \{0, 1\}}\}\) from $A$ on behalf of corrupt party $P_i$. Next, for each $i \in [n] \setminus H$, $w \in A_i$, $\gamma, \delta \in \{0, 1\}$, $\text{Sim}$ extracts $b_{w,\gamma,\delta} := E_2(\text{state}_{w,\gamma,\delta}, c_{w,\gamma,\delta})$.

- **Completing the execution with $\text{Sim}_\Phi$:** The interaction with $\text{Sim}_\Phi$ is completed assuming that the corrupt parties behave semi-honestly in this phase of the protocol. For a round in which a corrupt party is the speaker, $\text{Sim}$ sends the bit $b_{w,\gamma,\delta}$ to $\text{Sim}_\Phi$, where $\gamma$ and $\delta$ are chosen appropriately based on the state vector maintained and $\phi_w$. Let $Z \in \{0, 1\}^T$, where $Z_t$ represents the bit sent in the $t$th round of the computation phase of $\Phi$, be the output of this execution. And let $s^* \in \{0, 1\}^{L+T}$ be the state value at the end of execution. Also, for each $i \in H$, $t \in A_i$, $\gamma, \delta \in \{0, 1\}$, set $d^i_{w,\gamma,\delta} := d^i_{w,\gamma,\delta}$.

- **Round-2 messages from $\text{Sim}$ to $A$:** For each $i \in H$, the simulator $\text{Sim}$ generates the second round message on behalf of $P_i$ as follows:
MPC from R3PO

Let $\Phi$ be a conforming protocol for the same functionality, $\text{Com}$ be a UC-secure commitment scheme, and $\mathcal{O}^{\text{Com},i}_{\text{MPC}}$ be an R3PO scheme w.r.t. $(G_{\text{Com},i}^{\text{MPC}}, Q_{\text{Com},i}^{\text{MPC}}, R_{\text{MPC}})$.

Common Random String: For each $w \in \{L : T\}, \gamma, \delta \in \{0, 1\}$, sample $\text{crs}_{w,\gamma,\delta} \leftarrow \text{Com.\ Setup}(1^\kappa)$ and output $\{\text{crs}_{w,\gamma,\delta}\}_{w \in \{L : T\}, \gamma, \delta \in \{0, 1\}}$ as the common random string $\text{crs}$.

Round-1: Each party $P_i$ does the following:

- Compute $(z_i, \{G_w\}_{w \in A_i}) \leftarrow \text{pre}(1^\kappa, i, x_i)$.
- For every $w \in A_i, \gamma, \delta \in \{0, 1\}$ $(c_{w,\gamma,\delta}, d_{w,\gamma,\delta}) \leftarrow \text{Com.\ Commit}(\text{crs}_{w,\gamma,\delta}, G_w(\gamma, \delta))$.
- Send $(z_i, \{c_{w,\gamma,\delta}\}_{w \in A_i, \gamma, \delta \in \{0, 1\}})$ to every other party.

Round-2: Each party $P_i$ does the following:

- Set $s^0 := z_1 \parallel \ldots \parallel z_n$.
- Set $c := \{c_{w,\gamma,\delta}\}_{w \in \{L : T\}, \gamma, \delta \in \{0, 1\}}$.
- Set $\beta := \{d_{w,\gamma,\delta}\}_{w \in A_i, \gamma, \delta \in \{0, 1\}}$.
- Define a function $\mu_i^{(\beta)}$ over $\Sigma_{\text{MPC}}$ s.t.

$$
\mu_i^{(\beta)}(\sigma_{w-L}) = \begin{cases} 
\text{otherwise.} & \\
\text{where } \phi_w = (i, u, v), \gamma = s_u, \delta = s_v, & 
\end{cases}
$$

for all $w \in \{L : T\}$.
- Define $\alpha := \{G_w\}_{w \in A_i}$.
- Set $\rho_i \leftarrow \mathcal{O}^{\text{Com},i}_{\text{MPC}}(\pi^{(\alpha)}_\Phi, \sigma^{(\beta)}_\Phi, \mu_i^{(\beta)}, I_{\text{MPC}})$.
- Send $\rho_i$ to every other party.

Evaluation: To compute the output of the protocol, each party does the following:

- Set $s := z_1 \ldots || z_n$.
- Set $\overline{x}_j = ()$ for every $j \in [n]$.
- For every $w \in \{L : T\}$, do:
  * Parse $\phi_w = (i, u, v)$.
  * Add $\epsilon$ to the sequence $\overline{x}_w$.
  * Evaluate $\rho_i(\overline{x}_w)$ to get $d$.
  * Append the bit $\text{Com.\ Open}(\text{crs}_{w,\gamma,\delta}, (c_{w,\gamma,\delta}, d))$ to $s$, where $\gamma = s_u, \delta = s_v$.
  * For every $j \in [n] \setminus \{i\}$, add $\epsilon$ to the sequence $\overline{x}_j$.
- For every $w \in \{T : K\}$ s.t. $\phi_w = (i, u, v)$, append the bit $G_w(s_u, s_v)$ to $s$.
- Output the string $s[T : K]$.

**Fig. 30.** Our MPC construction
1. Set \( c := \{ c_{w,\gamma,\delta} \}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}} \).
2. Run \( c_{\text{MPC}}^{\text{Com},i} \) internally by simulating an interaction between an \( H_{\text{MPC}}^{\text{Com},i} \) and some \( Q \); where these are described below.
3. First, \( Q \) sends \( \{ \text{crs}_{w,\gamma,\delta} \}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}} \) to \( H_{\text{MPC}}^{\text{Com},i} \). The generator \( H_{\text{MPC}}^{\text{Com},i} \) then sends \( (z_i, \{ c_{w,\gamma,\delta} \}_{w \in A_i, \gamma, \delta \in \{0,1\}}) \) to \( Q \), while \( Q \) sends back the message \( \{ (z_j, \{ c_{w,\gamma,\delta} \}_{w \in A_j, \gamma, \delta \in \{0,1\}}) \}_{j \in [n] \setminus \{i\}} \). Set \( Q \) such that its reach is described by the vector \( x = Z \). Since \( c_{\text{MPC}}^{\text{Com},i} \) is allowed to look inside \( Q \), it finally outputs \((a_Q, \Pi_{\Phi,x}[s_{\text{crs}}^0], \{x\})\).
4. For each \( w \in A_i \) s.t. \( \phi_w = (i, u, v) \), set \( \gamma_w = s_w^* \) and \( \delta_w = s_w^* \).
5. Set \( \rho_i \leftarrow \text{Sim}_{\text{MPC}}^{\text{Com},i}(\pi^{(i)}_{\Phi,\text{crs},c,i}[s_{\text{crs}}^0], \mu^{(i)}_i, \mathcal{I}_{\text{MPC}}, \Pi_{\Phi,x}[s_{\text{crs}}^0], \{x, (x_{w-L}, d_w, \gamma_w, \delta_w) \}_{w \in A_i}) \).
6. Send \( \rho_i \) to \( A \).

- **Round-2 messages from \( A \) to Sim:** For every \( i \in [n] \setminus H \), Sim obtains the second round message \( \rho_i \) from \( A \) on behalf of the malicious party \( P_i \). The simulator halts with appropriate output to the ideal functionality.

**Proof of Indistinguishability.** We prove indistinguishability via a sequence of \(|H|+2\) hybrids. We describe the hybrids below.

- \( H_{\text{REAL}} \): This hybrid is the same as the real world execution.
- \( H_{\text{REAL}}' \): In this hybrid, we change the distribution of the common random string. In particular, common random strings for the corrupt parties are generated as in the simulation i.e., via the use of \( \mathcal{E}_1 \). Moreover, \( \mathcal{E}_2 \) is used to extract the bits from these commitments. Indistinguishability between hybrids \( H_0 \) and \( H_{\text{REAL}} \) follows by the binding property of Com.
- \( H_0 \): In this hybrid, we change the distribution of how the first round messages and the common random strings are generated for the honest parties. In particular, \( \text{Sim}_{E_0} \) is used to generate crs and the commitment \( c \) for every round \( w - L \) in which an honest party is the speaker and for every \( \gamma, \delta \in \{0,1\} \).
Moreover, the corresponding decommitments generated by \( \text{Sim}_{E_0} \) are hard-coded in the message function, inside the obfuscated program (the second-round message) for the corresponding honest party. In-distinguishability between hybrids \( H_0 \) and \( H_1 \) follows by the equivocal commitment security.
- \( H_i \) (for \( i \in H \)): In this hybrid, we change the distribution of the second round message for the honest party \( P_i \). In particular, we use \( c_{\text{MPC}}^{\text{Com},i} \) and \( \text{Sim}_{\text{MPC}}^{\text{Com},i} \) (just as in the simulation) to output the obfuscated program \( \rho_i \).
For this, the simulator Sim runs a mental execution of \( \Phi \) using the honest parties’ actual inputs and the extracted bits from \( \mathcal{E}_2 \) for the outputs of the corrupt parties. At the end, it learns the transcript \( Z \) and the state vector \( s^* \), which are then used to provide the necessary inputs to \( \text{Sim}_{\text{MPC}}^{\text{Com},i} \). Indistinguishability between two such consecutive hybrids follows from R3PO security of \( C_{\text{MPC}}^{\text{Com},i} \).
- \( H_{\text{REAL}}'' \): In this hybrid, we change how the transcript \( Z \), the honest party messages \( \{ z_i \}_{i \in H} \), random coins of corrupt parties and the state value \( s^* \) are generated. In particular, these values are generated as in the simulation i.e., via invoking \( \text{Sim}_\Phi \). Indistinguishability between \( H_{\text{IDEAL}}' \) and \( H_{\text{REAL}}'' \) follows from the malicious security of the underlying conforming protocol \( \Phi \). Finally, note that this is the same as the ideal world execution.

\[ \square \]