R3PO: Reach-Restricted Reactive Program Obfuscation and its Application to MA-ABE

Kaartik Bhushan[1], Sai Lakshmi Bhavana Obbattu[2], Manoj Prabhakaran[1], and Rajeev Raghunath[1]

[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). This is in stark contrast to the existing constructions for MA-ABE, which rely on the random oracle model and/or support 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, 30]. 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 [25], 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, 44, 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, 26, 30]. Our definition of R3PO allows an intuitive description of prior constructions like IBE [22] (with an easy extension to Identity-Based Functional Encryption [60]), 2-Round MPC [7, 30], and RBE [26], 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 grabled 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,[3] and in return obtain a strong privacy guarantee. Though the above transformation shows that standard MA-ABE

[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.

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, 57]). Like R3PO, many of these require security only when the program being obfuscated is generated appropriately [5, 8, 36, 57].

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, 53, 58], 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 ciruits were conceived by Yao [59]. The techniques of chaining multiple garbled circuits appeared in garbled RAM schemes [27, 28, 31, 46], and later several results like Laconic OT [18], IBE from DDH [21, 22], 2-round MPC [7, 30], 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) [11, 32] and pairing based assumptions [33, 37, 43]. But for MA-ABE, first proposed in [17], realizations so far have been limited. In the standard GID model, [44] 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 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. More recently, [56] gave a (current state-of-the-art) construction for MA-ABE in the plain model for subset policies (including DNF formulae) from the new evasive LWE assumption. Their construction however requires a global setup.

[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 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 $f_{\mathsf{id}}(m)$. 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 [60], 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, 30], 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 π and a message function μ, so that, on reaching a state σ, 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, μ is arbitrary (and secret), and a public π is as specified below.

– **Commitment.** π_c incorporates a commitment string c. On input d at the start state, if d decommits c to m, then π_c transitions to a state σ_m encoding m.

– **Signature.** A signature verification key vk is encoded in the start state σ_{vk} of π (denoted as $\pi[\sigma_{\mathsf{vk}}]$), from where, given a valid signature on a message m as input, it transitions to a state σ_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 π in the two cases.) In these examples, π is not hidden, and the goal of obfuscating such a program would be to hide μ. More generally, π and μ can both have secrets in them (when defining reactive programs formally, we will denote them as $\pi^{(\alpha)}$ and $\mu^{(\beta)}$, where α and β 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-extaction and the rest of the simulation below.

Reach Extractability. Which states in a program π 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 Π that will reach all the states reachable by the adversary, and perhaps more. Here we allow the extractor to specify Π, which belongs to a transition function family $\mathring{\mathcal{P}}$ that may be different from the transition program family $\mathcal{P}$ 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 π_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 π_c. Instead, we allow the extractor to specify a different program Π_m which accepts m itself as the input and transitions to σ_m.

This extractor is reach bounding, because, due to the binding property of the commitment scheme, the only state Q could reach in π_c is also σ_m.

– **Signature:** In this case, G internally samples a pair $(\mathsf{sk}, \mathsf{vk})$ 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 $\Pi = \pi[\sigma_{\mathsf{vk}}]$. Note that here the program family to be used by the extractor $\mathring{\mathcal{P}}$ is the same as the one being obfuscated $\mathcal{P}$ (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 Π and inputs to it).

– Stage 2: Given the output of the original message function μ 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 incoporate a corresponding *reach restriction* requirement into our definition of reach extractability of a reactive program (Definition 15), 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 π 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 Σ_1 consist only of the start state and Σ_2 consist of all states of the form σ_m. Since the extractor outputs only one message m, the reach restriction property already holds.

– **Signature:** We let Σ_1 consist of all the potential start states σ_{vk} and Σ_2 consist of all states of the form σ_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 restriction in Π must correspond to forging signatures with respect to the key sampled by G. This is enforced by keeping vk in the start state of $\pi[\mathsf{vk}]$ (rather than in the transition function itself), which in turn forces Π 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 $\pi \in \mathcal{P}$, over a state space $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_N$ as consisting of separate programs $\widehat{\pi}_1, \ldots, \widehat{\pi}_N$, such that $\widehat{\pi}_i$ is identical to π on states $\sigma \in \Sigma_i$, and in other states it ignores all inputs (i.e., remains at the same state). Let $\mathcal{P}_i$ denote the class of such programs $\widehat{\pi}_i$. W.l.o.g. (due to reach-restriction), we require π to not have any transitions between states in the same part, and hence each $\widehat{\pi}_i$ 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 $\mathcal{P}$ from obfuscators for $\mathcal{P}_i$. However, this runs into an immediate problem: When executing a program in $\mathcal{P}$, the state reached in Σ_i is dynamically determined by the inputs used, whereas when obfuscating a program in $\mathcal{P}_i$, its start state needs to be fixed. The resolution of this conundrum, which goes back to [18, 21, 22, 27, 28, 31, 46], is to provide a **garbled circuit** that can dynamically compute the obfuscation of $\widehat{\pi}_i[\sigma_i]$ with the correct start state σ_i; the input to this garbled circuit would be the labels encoding σ_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

[4] Formally, we do not define R3P, but only an R3P Generator, as a program generator (Definition 14) that has a reach extractor.

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 $\boxed{\boxed{G}\ J}$ (denoting that J internally runs G as a black box) and $\boxed{Z\ \boxed{H}}$ are indistinguishable from each other from the point of view of any adversary Q (or more precisely, for $\boxed{Q\ \boxed{W}}$, 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 $\widehat{\pi}$ produced by H corresponds to a single step in the program π produced by G. More precisely, the state space of the generator H corresponds to a *part* Σ_i of the state space of G, and we require that the start state of $\widehat{\pi}$ is the same as the only state in Σ_i that is reachable in π.

Now, by requiring this, we require J to know the reachable state in π 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 Σ_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 $\mathcal{O}_i$, 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 Σ_i, such that on reaching $\sigma \in \Sigma_i$, an evaluator would have the labels that encode σ as input for GC_i, and GC_i would then output $\mu(\sigma)$ as well as an obfuscation $\mathcal{O}_i(\widehat{\pi}_i[\sigma], \widehat{\mu}_i)$ (using a hard-coded random tape). Feeding an input x to this obfuscated program will release the labels for the state $\widehat{\pi}_i(\sigma, x) = \pi(\sigma, x)$.

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 $\mathcal{O}_i(\widehat{\pi}_i[\sigma], \widehat{\mu}_i)$, 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 $\boxed{\boxed{G}\ J_i}$ to $\boxed{Z_i\ \boxed{H_i}}$, 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 $\boxed{Z_i\ \boxed{H_i}}$ to $\boxed{\boxed{G}\ J_i}$ 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 Appendix B.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.[5] We instantiate a puncturable signature scheme and give an R3PO scheme for this program family assuming an OTSE scheme in Appendix B.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).

– ϵ-*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. ϵ-transitions provide the essential syntactic sugar to enable this. R3PO for an ϵ-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 and elaborated in Appendix D. 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.

– We start by identifying a reactive program family, such that an R3PO for it directly yields our application.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.
– We consider the one-step restrictions of this reactive program as another reactive program family, and carry out the following two steps:
  - 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.
  - 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).
– Then we *simply invoke our composition theorem* to obtain an R3PO for the original program family.
– 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 B.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).

[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.

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, wherein 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. One such example is the 2-round MPC protocol of [7, 30], which we rederive in Appendix E using R3PO for commitment opening. In this construction, as sketched in Section 2.1, several programs obfuscated using R3PO are involved. The security of R3PO can be used to move to an "ideal" execution of the 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 $A_1, \ldots, A_{\mathsf{N}}$), a sender and a receiver. The algorithms in an 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 $(\mathsf{mpk}_i, \mathsf{msk}_i)$ 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 $\mathsf{req} = (\mathsf{req}_1, \ldots, \mathsf{req}_{\mathsf{N}})$ for a global identifier gid and attribute set $\bar{\mathsf{x}}$ from the public keys. It can then submit a key-request query (of the form $(\mathsf{gid}, \mathsf{req}_i)$) to an authority A_i and get back a key-component $\mathsf{sk}_{\mathsf{gid},\mathsf{req}_i}$ from A_i (req will hide $\bar{\mathsf{x}}$).
- Key-Gen: an authority A_i receives as input a key-request req_i for a global identifier gid, and outputs a key-component $\mathsf{sk}_{\mathsf{gid},\mathsf{req}_i}$ that incorporates an attribute-granting policy Θ_i^{gid} (which, for simplicity, we do not consider a secret).
- Encryption: a sender can encrypt a message m with a ciphertext policy ϕ, using the public keys of the authorities to produce a ciphertext $\mathsf{ct}_{m,\phi}$.
- Decryption: a receiver can decrypt a ciphertext $\mathsf{ct}_{m,\phi}$ using key components of the form $(\mathsf{sk}_{\mathsf{gid},\mathsf{req}_1}, \cdots, \mathsf{sk}_{\mathsf{gid},\mathsf{req}_{\mathsf{N}}})$ where all the requests req_i were generated using gid and $\bar{\mathsf{x}}$ such that $\Theta_i^{\mathsf{gid}}(\bar{\mathsf{x}}) = 1$ for all i, and $\phi(\bar{\mathsf{x}}) = 1$.

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.[6]

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 $\bar{\mathsf{x}}$ to the authorities, even if all of them collude. When the receiver is corrupt, we guarantee that, for any choice of (ϕ, m_0, m_1), the adversary cannot distinguish between the encryptions of m_0 and m_1 w.r.t. a policy ϕ, unless for a pair $(\mathsf{gid}, \bar{\mathsf{x}})$ such that $\Theta_i^{\mathsf{gid}}(\bar{\mathsf{x}}) = 1$ for *all* honest authorities A_i, $\phi(\bar{\mathsf{x}}) = 1$ and the adversary sent a valid key request for $(\mathsf{gid}, \bar{\mathsf{x}})$ (i.e., a request that can be

[6] 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.

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 req_i is a commitment to $\bar{\mathsf{x}}$ (using a common random string in the public-key of A_i). 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 $\bar{\mathsf{x}}$ to gid, and if so, issue standard CP-ABE keys for $\bar{\mathsf{x}}$. More precisely, the reactive program $(\pi^{(\alpha)}, \mu^{(\beta)})$ works as follows (with A_i's CP-ABE master secret-key constituting the secret β; α can be empty, or alternately, can be used to store Θ_i^{gid} privately):

- at the start state accepts a decommitment for req_i and transitions to a state with $\bar{\mathsf{x}}$. There, if $\Theta_i^{\mathsf{gid}}(\bar{\mathsf{x}}) = 1$, then it outputs a signature on $(\mathsf{gid}, \bar{\mathsf{x}})$, using A_i's signature key.
- Then, it moves through $\mathsf{N} - 1$ states accepting signatures on $(\mathsf{gid}, \bar{\mathsf{x}})$ from all the other servers.
- On reaching the last of these states, it outputs a CP-ABE key for the attribute $\bar{\mathsf{x}}$, under a (standard) CP-ABE scheme for which A_i is the authority.

If $\Theta_i^{\mathsf{gid}}(\bar{\mathsf{x}}) = 1$ for all i, then the receiver can obtain the CP-ABE keys for $\bar{\mathsf{x}}$ 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.

Note that if even one (honest) authority's key component is missing, no honest authority's CP-ABE key can be obtained. This is crucial because the CP-ABE keys do not involve gid and cannot prevent the use of keys obtained using multiple gids.

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 $\mathcal{O}$ is an R3PO scheme for both the generators.)

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 [25] 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 [10, 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, 26, 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

[36] or compute-and-compare obfuscation based on LWE [57]. R3PO could be considered to be in the latter group, but with a much richer class of applications compared to the others.

The original notions of obfuscation require worst-case security, but there are several others, including obfuscation of evasive circuit families [5], strong iO [8], reencryption obfuscation [36], compute-and-compare obfuscation [57], etc. which require only *distributional security*, when the program being obfuscated is sampled from distributions with particular properties. Again, R3PO falls into the latter class here, with the sampling process being interactive.

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, 29] 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 Preliminaries

3.1 Garbling Scheme

Definition 1 (Garbled Circuits). A Garbling scheme is a pair of PPT algorithms $(\mathsf{GCkt}, \mathsf{GCEval})$ such that

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

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

$$\Pr[C(x) = \mathsf{GCEval}(\mathrm{GC}, lab_x)] = 1$$

where $\mathrm{GC} \leftarrow \mathsf{GCGarble}(1^\kappa, C, \widehat{\beta})$ and x_w corresponds to the value on input wire w(while evaluating C) and $lab_x = \{\widehat{\beta}^{w,x_w}\}_{w \in \mathsf{inputwire}(C)}$.

Security. For security, we require that

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

   $$\mathrm{GC}, lab_x \approx \mathsf{GSim}(\Phi(C), lab_x, y)$$

   where $\mathrm{GC} \leftarrow \mathsf{GCGarble}(C, \widehat{\beta})$, $lab = \{\widehat{\beta}^{w,x_w}\}_{w \in \mathsf{inputwire}(C)}$ $y = C(x)$ and $\Phi(C)$ denotes the topology of the circuit C.
2. There exists a simulator $\mathsf{GCircSim}$ such that for any circuit C and uniformly random $\widehat{\beta}$, we have that

   $$\mathrm{GC} \approx \mathsf{GCircSim}(\Phi(C))$$

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

◁

3.2 CP-ABE

Definition 2 (Ciphertext-Policy ABE (ABE)). An ABE scheme for a message space M, attribute space $\{0,1\}^l$ and a circuit class $\mathcal{C}$ consists of PPT algorithms defined as follows:

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

The following correctness and security properties are required:

1. **Correctness**: $\forall \kappa$, messages $m \in \mathsf{M}$, policies $\phi \in \mathcal{C}$, attributes $x \in \{0,1\}^l$ s.t. $\phi(x) = 1$, it holds that if $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa, 1^l)$, $\mathsf{sk}_x \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x)$, then
$$\Pr\left[\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}_x, \mathsf{Encrypt}(\mathsf{mpk}, \phi, m)) = m\right] = 1 - \mathsf{negl}(\kappa)$$
2. **Selective Security**: For any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(.)$ such that the following holds
$$\Pr[\mathrm{IND}^{\mathrm{ABE}}_{\mathrm{SEL}}(\mathcal{A}) = 1] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$$
where $\mathrm{IND}^{\mathrm{ABE}}_{\mathrm{SEL}}$ is shown in 1.

Experiment $\mathbf{IND}^{\mathbf{ABE}}_{\mathbf{SEL}}$

Parameter: Let κ be the security parameter.

- $(s_{\mathcal{A}_0}, \phi) \leftarrow \mathcal{A}_0(1^\kappa)$.
- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa, 1^l)$.
- $(m_0, m_1, s_{\mathcal{A}_1}) \leftarrow \mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{msk},.)}(s_{\mathcal{A}_0}, \mathsf{mpk})$.
- $b \leftarrow \{0,1\}$.
- $m := m_b$.
- $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \phi, m)$.
- $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},.)}(s_{\mathcal{A}_1}, \mathsf{mpk}, \mathsf{ct})$.
- If $\mathcal{A}_1$ or $\mathcal{A}_2$ 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. 1. ABE ind-security Experiment.

3. **Full Security**: For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(.)$ such that the following holds
$$\Pr[\mathrm{IND}^{\mathrm{ABE}}_{\mathrm{FULL}}(\mathcal{A}) = 1] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$$
where $\mathrm{IND}^{\mathrm{ABE}}_{\mathrm{FULL}}$ is shown in 2.

◁

Experiment $\mathbf{IND}^{\mathbf{ABE}}_{\mathbf{FULL}}$

Parameter: Let κ be the security parameter.

- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa, 1^l)$.
- $(s_{\mathcal{A}_0}, \phi) \leftarrow \mathcal{A}_0(1^\kappa)$.
- $(\phi, m_0, m_1, s_{\mathcal{A}_1}) \leftarrow \mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{msk},.)}(s_{\mathcal{A}_0}, \mathsf{mpk})$.
- $b \leftarrow \{0,1\}$.
- $m := m_b$.
- $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \phi, m)$.
- $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},.)}(s_{\mathcal{A}_1}, \mathsf{mpk}, \mathsf{ct})$.
- If $\mathcal{A}_1$ or $\mathcal{A}_2$ 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. 2. ABE full ind-security Experiment.

3.3 Commitment Schemes

Definition 3 (Commitment scheme). A UC-secure commitment scheme Com over input space $\mathsf{M} = \{0,1\}^l$ consists of the algorithms $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ with the following syntax:

- $\mathsf{Setup}(1^\kappa) \to \mathsf{crs}$: takes the security parameter κ as input (in unary), and outputs the common reference string crs.
- $\mathsf{Commit}(\mathsf{crs}, x) \to (c, d)$: takes as input a common reference string crs, a string x, and outputs a commitment c, decommitment d.
- $\mathsf{Open}(\mathsf{crs}, c, d) \to x$: takes as input a common reference string crs, a commitment c, a decommitment d, and outputs a string x.

and the following properties:

- **Hiding.** For all κ, all strings $x_0, x_1 \in \mathcal{X}$ and PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(.)$ such that if $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$, $(c_0, d_0) \leftarrow \mathsf{Commit}(\mathsf{crs}, x_0)$, $(c_1, d_1) \leftarrow \mathsf{Commit}(\mathsf{crs}, x_1)$, then it holds that

$$\left| \Pr\left[\mathcal{A}(\mathsf{crs}, c_0) = 1\right] - \Pr\left[\mathcal{A}(\mathsf{crs}, c_1) = 1\right] \right| \leq \mathsf{negl}(\kappa)$$

- **Binding.** There exists a PPT extractor $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$, such that for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, it holds that

$$\left| \Pr\left[\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\kappa) = 1\right] - \Pr\left[\mathrm{IND}^{\mathrm{IDEAL}}_{\mathcal{A},\mathcal{E}}(1^\kappa) = 1\right] \right| \leq \mathsf{negl}(\kappa)$$

where the experiments are defined as follows:

Experiment $\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\kappa)$:
$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$
$(\mathsf{st}_{\mathcal{A}_1}, c, d) \leftarrow \mathcal{A}_1(\mathsf{crs})$
$x \leftarrow \mathsf{Open}(\mathsf{crs}, c, d)$
Output $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}_1}, x)$

Experiment $\mathrm{IND}^{\mathrm{IDEAL}}_{\mathcal{A},\mathcal{E}}(1^\kappa)$:
$(\mathsf{crs}, \mathsf{st}_{\mathcal{E}_1}) \leftarrow \mathcal{E}_1(1^\kappa)$
$(\mathsf{st}_{\mathcal{A}_1}, c, d) \leftarrow \mathcal{A}_1(\mathsf{crs})$
$x \leftarrow \mathcal{E}_2(\mathsf{st}_{\mathcal{E}_1}, c)$
Output $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}_1}, x)$

– **Equivocal Commitment.** There exists PPT simulator Sim_{Eq} such that for any $x \in \mathcal{X}$, it holds that

$$\left\{(\mathsf{crs}, (c, d_x)) : (\mathsf{crs}, c, \{d_y \mid y \in \mathcal{X}\}) \leftarrow \mathsf{Sim}_{Eq}(1^\kappa)\right\} \approx \left\{(\mathsf{crs}, \mathsf{Commit}(\mathsf{crs}, x)) : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)\right\}$$

◁

Statistical v.s. Computational Binding. The above definition requires statistical binding, where each commitment c has at most a single string x that can be opened to (and extracted by $\mathcal{E}$). We also define a weaker definition for a semi-honest adversary that requires only computational binding, where an honestly generated commitment c could have multiple openings, but it must be hard for an adversary to open it to anything other than x.

Definition 4 (Weak Commitment scheme). A weak commitment scheme Com over input space $\mathsf{M} = \{0,1\}^l$ consists of the algorithms $(\mathsf{Commit}, \mathsf{Open})$ with the following syntax:

– $\mathsf{Commit}(x) \to (c, d)$: takes as input a string x, and outputs a commitment c, decommitment d.

– $\mathsf{Open}(c, d) \to x$: takes as input a commitment c, a decommitment d, and outputs a string x.

and the following properties:

– **Hiding.** For all κ, all strings $x_0, x_1 \in \mathcal{X}$ and PPT adversaries $\mathcal{A}$, there exists a negligible function negl (.) s.t. if $(c_0, d_0) \leftarrow \mathsf{Commit}(x_0)$, $(c_1, d_1) \leftarrow \mathsf{Commit}(x_1)$, then it holds that

$$\left|\Pr\left[\mathcal{A}(c_0) = 1\right] - \Pr\left[\mathcal{A}(c_1) = 1\right]\right| \leq \mathsf{negl}(\kappa)$$

– **Weakly-Binding.** For all κ, all strings $x \in \mathcal{X}$ and PPT adversaries $\mathcal{A}$, the following probability is negligible:

$$\Pr\left[(c, d) \leftarrow \mathsf{Commit}(x; r),\ m' \leftarrow \mathsf{Open}(c, \mathcal{A}(x, r)),\ m' \notin \{m, \perp\}\right]$$

◁

3.4 Oblivious Transfer

Definition 5 (Oblivious Transfer (OT)). A UC-secure OT scheme consists of four algorithms $(\mathsf{Setup}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ with the following syntax:

– $\mathsf{Setup}(1^\kappa) \to \mathsf{crs}$: takes as input the security parameter 1^κ, and outputs the common random string crs.

– $\mathsf{OT}_1(\mathsf{crs}, b) \to (\mathsf{ots}_1, \omega)$: takes as input a common random string crs, bit b, and outputs a tuple (ots_1, ω), where ots_1 is a ciphertext and ω is a secret state.

– $\mathsf{OT}_2(\mathsf{crs}, \mathsf{ots}_1, (m_0, m_1)) \to \mathsf{ots}_2$: takes as input a common random string crs, a ciphertext ots_1, messages m_0, m_1, and outputs a ciphertext ots_2.

– $\mathsf{OT}_3(\mathsf{crs}, \mathsf{ots}_2, \omega) \to m_b$: takes as input a common random string crs, a ciphertext ots_2, a secret state ω, and outputs a message m_b.

and the following properties:

– **Correctness.** It holds for all κ, every bit $b \in \{0,1\}$, every string ω, every pair of messages m_0, m_1, that if $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$, $\mathsf{ots}_1 \leftarrow \mathsf{OT}_1(\mathsf{crs}, b)$ and $\mathsf{ots}_2 \leftarrow \mathsf{OT}_2(\mathsf{crs}, \mathsf{ots}_1, (m_0, m_1))$, then

$$\mathsf{OT}_3(\mathsf{crs}, \mathsf{ots}_2, \omega) = m_b$$

– **Receiver's Security.** It holds for all κ that

$$\left\{(\mathsf{crs}, \mathsf{ots}_1) : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa), (\mathsf{ots}_1, \omega) \leftarrow \mathsf{OT}_1(\mathsf{crs}, 0)\right\} \approx$$
$$\left\{(\mathsf{crs}, \mathsf{ots}_1) : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa), (\mathsf{ots}_1, \omega) \leftarrow \mathsf{OT}_1(\mathsf{crs}, 1)\right\}$$

– **Sender's Security.** There exists a PPT extractor $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ such that for any choice of messages $K_0, K_1 \in \{0,1\}^\kappa$, and PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have that

$$\left|\Pr[\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\kappa, K_0, K_1) = 1] - \Pr[\mathrm{IND}^{\mathrm{IDEAL}}_{\mathcal{A},\mathcal{E}}(1^\kappa, K_0, K_1) = 1]\right| \leq \mathsf{negl}(\kappa)$$

Experiment $\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\kappa, K_0, K_1)$:
$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$
$(\mathsf{st}_{\mathcal{A}_1}, \mathsf{ots}_1) \leftarrow \mathcal{A}_1(\mathsf{crs})$
$\mathsf{ots}_2 \leftarrow \mathsf{OT}_2(\mathsf{crs}, \mathsf{ots}_1, K_0, K_1)$
Output $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}_1}, \mathsf{ots}_2)$

Experiment $\mathrm{IND}^{\mathrm{IDEAL}}_{\mathcal{A},\mathcal{E}}(1^\kappa, K_0, K_1)$:
$(\mathsf{crs}, \tau) \leftarrow \mathcal{E}_1(1^\kappa)$
$(\mathsf{st}_{\mathcal{A}_1}, \mathsf{ots}_1) \leftarrow \mathcal{A}_1(\mathsf{crs})$
$\beta \leftarrow \mathcal{E}_2(\tau, \mathsf{ots}_1)$
$L_0 := K_\beta$ and $L_1 := K_\beta$
$\mathsf{ots}_2 \leftarrow \mathsf{OT}_2(\mathsf{crs}, \mathsf{ots}_1, L_0, L_1)$
Output $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}_1}, \mathsf{ots}_2)$

– **Equivocal Receiver's Security.** There exists a PPT simulator Sim_{Eq} such that for any $b \in \{0,1\}$, it holds that

$$\left\{(\mathsf{crs}, (\mathsf{ots}_1, \omega_b)) : (\mathsf{crs}, \mathsf{ots}_1, \omega_0, \omega_1) \leftarrow \mathsf{Sim}_{Eq}\right\} \approx \left\{(\mathsf{crs}, \mathsf{OT}_1(\mathsf{crs}, b)) : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)\right\}$$

◁

3.5 Hash Schemes

Definition 6 (Block-Openable CRH). A block-openable collision-resistant hash scheme over input space $\mathcal{X} = \{0,1\}^n$ and digest space $\mathcal{D} = \{0,1\}^d$ consists of the algorithms (crsGen, hash, $\mathsf{openBlock}$, $\mathsf{acceptBlock}$) with the following syntax:

– $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa)$. It is a probabilistic algorithm that takes as input the security parameter 1^κ and outputs a common reference string crs.
– $\mathsf{digest} \leftarrow \mathsf{hash}(\mathsf{crs}, m)$. It is a deterministic algorithm that takes as input a common reference string crs and a message m, and outputs a digest.
– $w \leftarrow \mathsf{openBlock}(\mathsf{crs}, m, i)$. It is a deterministic algorithm that takes as input a common reference string crs, a message m and a block index $i \in [n]$ s.t. $m = m_1 \| \ldots \| m_n$, and outputs a string w.
– $y \leftarrow \mathsf{acceptBlock}(\mathsf{crs}, \mathsf{digest}, i, w)$. 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 $m \in \mathsf{M}$, and all crs in the support of crsGen,

$$\mathsf{acceptBlock}(\mathsf{crs}, \mathsf{hash}(\mathsf{crs}, m), i, \mathsf{openBlock}(\mathsf{crs}, m, i)) = m_i,$$

where $i \in [n]$ is a block index and $m = m_1 \| \ldots \| m_n$.
– **Collision Resistance:** For all PPT adversaries $\mathcal{A}$, the following probability is negligible:

$$\Pr_{\substack{\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa) \\ (m,i,w) \leftarrow \mathcal{A}(\mathsf{crs})}} \left[\mathsf{acceptBlock}(\mathsf{crs}, \mathsf{hash}(\mathsf{crs}, m), i, w) \notin \{\perp, m_i\}\right].$$

◁

Compression Factor. The compression factor is given by $\frac{n}{d}$. For example, a scheme that hashes inputs of size $2d$ to a digest of size d corresponds to a factor-2 compression. We would typically require hash and $\mathsf{openBlock}$ to produce an output that is much shorter than the messages being hashed, that is $d << n$.

3.6 Laconic Oblivious Transfer

Definition 7 (Laconic OT). A laconic OT (ℓOT) scheme syntactically consists of four algorithms crsGen, hash, Send and $\mathsf{Receive}$.

- $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa)$. It takes as input the security parameter 1^κ and outputs a common reference string crs.
- $(\mathsf{digest}, \widehat{\mathsf{D}}) \leftarrow \mathsf{hash}(\mathsf{crs}, D)$. It takes as input a common reference string crs and a database $D \in \{0,1\}^*$ and outputs a digest digest of the database and a state $\widehat{\mathsf{D}}$.
- $\mathsf{ct} \leftarrow \mathsf{Send}(\mathsf{crs}, \mathsf{digest}, L, m_0, m_1)$. It takes as input a common reference string crs, a digest digest, a database location $L \in \mathbb{N}$ and two messages m_0 and m_1 of length κ, and outputs a ciphertext ct.
- $m \leftarrow \mathsf{Receive}^{\widehat{\mathsf{D}}}(\mathsf{crs}, \mathsf{ct}, L)$. This is a RAM algorithm with random read access to $\widehat{\mathsf{D}}$. It takes as input a common reference string crs, a ciphertext ct, and a database location $L \in \mathbb{N}$. 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 $(\mathsf{crsGen}, \mathsf{hash}, \mathsf{Send}, \mathsf{Receive})$.

- **Correctness:** We require that it holds for any database D of size at most $m = \mathsf{poly}(\kappa)$, for any polynomial function $\mathsf{poly}(\cdot)$, any memory location $L \in [M]$, and any pair of messages $(m_0, m_1) \in \{0,1\}^\kappa \times \{0,1\}^\kappa$ that

$$\Pr\left[m = m_{D[L]} \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa) \\ (\mathsf{digest}, \widehat{\mathsf{D}}) \leftarrow \mathsf{hash}(\mathsf{crs}, D) \\ \mathsf{ct} \leftarrow \mathsf{Send}(\mathsf{crs}, \mathsf{digest}, L, m_0, m_1) \\ m \leftarrow \mathsf{Receive}^{\widehat{\mathsf{D}}}(\mathsf{crs}, \mathsf{ct}, L) \end{array} \right] = 1,$$

  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 $\ell OT.\mathsf{Sim}$, such that for all PPT adversaries $\mathcal{A}$, the following holds

$$\text{REAL}_{\mathcal{A}}(1^\kappa) \approx \text{IDEAL}_{\mathcal{A}}^{\ell OT.\mathsf{Sim}}(1^\kappa)$$

  where the experiments are defined as follows:

Experiment $\text{REAL}_{\mathcal{A}}(1^\kappa)$:	**Experiment** $\text{IDEAL}_{\mathcal{A}}^{\ell OT.\mathsf{Sim}}(1^\kappa)$:
$\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa)$	$\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa)$
$(aux, m_0, m_1, D, L) \leftarrow \mathcal{A}(\mathsf{crs})$	$(aux, m_0, m_1, D, L) \leftarrow \mathcal{A}(\mathsf{crs})$
$\mathsf{digest} := \mathsf{hash}(\mathsf{crs}, D)$	
$\mathsf{ct} \leftarrow \mathsf{Send}(\mathsf{crs}, \mathsf{digest}, L, m_0, m_1)$	$\mathsf{ct}' \leftarrow \ell OT.\mathsf{Sim}(\mathsf{crs}, D, L, m_{D[L]})$
Output (aux, ct)	Output (aux, ct')

  In the above, database D should be of size at most $M = \mathsf{poly}(\kappa)$, for any polynomial function $\mathsf{poly}(\cdot)$, 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 κ independent of the size of the databse; we will assume for simplicity that $|\mathsf{digest}| = \kappa$. Moreover, the algorithm hash runs in time $|D|.\mathsf{poly}(\log|D|, \kappa)$, Send and $\mathsf{Receive}$ run in time $\mathsf{poly}(\log|D|, \kappa)$.

$\triangleleft$

3.7 Signature Schemes

Definition 8 (One-Time Signature). A one-time signature (OTS) scheme OTS for message space $\mathsf{M} = \{0,1\}^n$ consists of three algorithms as follows:

– $\mathsf{OTS.gen}(1^\kappa) \to (\mathsf{OTS.vk}, \mathsf{OTS.sk})$: On input security parameter κ, outputs a pair $(\mathsf{OTS.vk}, \mathsf{OTS.sk})$ of verification and signing keys.
– $\mathsf{OTS.sign}(\mathsf{OTS.sk}, x) \to \tau$: On input a signing key $\mathsf{OTS.sk}$ and message x, outputs a signature τ.
– $\mathsf{OTS.verify}(\mathsf{OTS.vk}, \tau, x) \to b'$: On input verification key $\mathsf{OTS.vk}$, signature τ and a message x, outputs a bit b'.

where they satisfy the following properties.

1. **Succinctness**: For $(\mathsf{OTS.vk}, \mathsf{OTS.sk}) \leftarrow \mathsf{OTS.gen}(1^\kappa)$ it holds that $|\mathsf{OTS.vk}|$ is independent of n and only depends on κ.
2. **Correctness of verification**: $\forall \kappa, x \in \mathsf{M}$, it holds that if $(\mathsf{OTS.vk}, \mathsf{OTS.sk}) \leftarrow \mathsf{OTS.gen}(1^\kappa)$, then

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

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

$$\Pr[\textsc{Sig-forge}^{1\text{-time}}(\mathcal{A}) = 1] \leq \mathsf{negl}(\kappa)$$

where $\textsc{Sig-forge}^{1\text{-time}}$ is shown in Figure 3.

Experiment $\textsc{Sig-forge}^{1\text{-time}}$

Parameter: Let κ be the security parameter.

– $(\mathsf{OTS.vk}, \mathsf{OTS.sk}) \leftarrow \mathsf{OTS.gen}(1^\kappa)$.
– $(m, \tau) \leftarrow A^{\mathsf{OTS.sign}(\mathsf{OTS.sk}, .)}(\mathsf{OTS.vk})$, where A can make a single query m' to oracle $\mathsf{OTS.sign}(\mathsf{OTS.sk}, .)$.
– If $\mathsf{OTS.verify}(\mathsf{OTS.vk}, \tau, m) = 1$ and $m \neq m'$, output 1.
Else, output 0.

Fig. 3. One-time Signature unforgeability security Experiment.

◁

Definition 9 (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:

– $\mathsf{OTSE.enc}(\mathsf{OTSE.vk}, i, b, m) \to \mathsf{ct}$: On input verification key $\mathsf{OTSE.vk}$, a bit position i, a bit b and plaintext m, outputs a ciphertext ct.
– $\mathsf{OTSE.dec}(\mathsf{OTSE.vk}, \tau, x, \mathsf{ct}) \to m$: On input verification key $\mathsf{OTSE.vk}$, message x, signature τ and ciphertext ct, outputs a plaintext m.

Where they satisfy the following properties.

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

$$\mathsf{OTSE.dec}(\mathsf{OTSE.vk}, \tau, x, \mathsf{OTSE.enc}(\mathsf{OTSE.vk}, i, x_i, m)) = m$$

2. **Selective security of encryption**: For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, there exists a negligible function $\mathsf{negl}(.)$ such that the following holds:

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

where IND^{OTSE} is shown in Figure 4.

Experiment IND^{OTSE}

Parameter: Let κ be the security parameter.

- $(s_{\mathcal{A}_1}, x) \leftarrow \mathcal{A}_1(1^\kappa)$.
- $(\mathsf{OTSE.vk}, \mathsf{OTSE.sk}) \leftarrow \mathsf{OTSE.gen}(1^\kappa)$.
- $\tau \leftarrow \mathsf{OTSE.sign}(\mathsf{OTSE.sk}, x)$.
- $(s_{\mathcal{A}_2}, i, m_0, m_1) \leftarrow \mathcal{A}_2(s_{\mathcal{A}_1}, \mathsf{OTSE.vk}, \tau)$.
- $b^* \leftarrow \{0, 1\}$.
- $m^* := m_{b^*}$.
- $c^* \leftarrow \mathsf{OTSE.enc}(\mathsf{OTSE.vk}, i, 1 - x_i, m^*)$.
- $b' \leftarrow \mathcal{A}_3(s_{\mathcal{A}_2}, c^*)$.
- Output 1 if $b' = b^*$ and 0 otherwise.

Fig. 4. One-time Signature with Encryption ind-security Experiment.

◁

Definition 10 (Signature scheme). A signature scheme Sig for message space $\mathsf{M} = \{0,1\}^n$ consists of three algorithms as follows:

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

where they satisfy the following properties.

1. **Succinctness**: For $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.gen}(1^\kappa)$ it holds that $|\mathsf{Sig.vk}|$ is independent of n and only depends on κ.
2. **Correctness of verification**: $\forall \kappa, x \in \mathsf{M}$, it holds that if $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.gen}(1^\kappa)$; then

$$\mathsf{Sig.verify}(\mathsf{Sig.vk}, \mathsf{Sig.sign}(\mathsf{Sig.sk}, x), x) = 1$$

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

$$\Pr[\text{SIG-forge}(\mathcal{A}) = 1] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$$

where SIG-forge is shown in Figure 5.

Experiment SIG-forge

Parameter: Let κ be the security parameter.

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

Fig. 5. Signature unforgeability security Experiment.

◁

Definition 11 (Puncturable Signature). A puncturable signature scheme Sig is a signature scheme that consists of the following additional algorithms:

- $\mathsf{Sig.punct}(\mathsf{Sig.sk}, \mathsf{mpre}) \rightarrow \mathsf{Sig.sk}'$: On input a signing key $\mathsf{Sig.sk}$ and a message prefix mpre, outputs a punctured signing key $\mathsf{Sig.sk}'$.
- $\mathsf{Sig.psign}(\mathsf{Sig.sk}', m) \rightarrow \tau$: On input a punctured signing key $\mathsf{Sig.sk}'$ and a message m, outputs a signature τ.

where they satisfy the following properties.

1. **Correctness of verification**: $\forall \kappa$, prefix mpre and $x \in \mathsf{M}$, it holds that if $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.gen}(1^\kappa), \mathsf{Sig.sk}' \leftarrow \mathsf{Sig.punct}(\mathsf{Sig.sk}, \mathsf{mpre})$, and x does not have mpre as prefix, then
$$\mathsf{Sig.verify}(\mathsf{Sig.vk}, \mathsf{Sig.psign}(\mathsf{Sig.sk}', x), x) = 1$$
2. **Unforgeability**: For any PPT adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$, there exists a negligible function $\mathsf{negl}(.)$ such that the following holds:
$$\Pr[\text{SIG-forge}^{\text{PUNCT}}(\mathcal{A}) = 1] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$$
where $\text{SIG-forge}^{\text{PUNCT}}$ is shown in Figure 6.

◁

3.8 Puncturable PRF

Definition 12 ((Prefix) Puncturable Pseudorandom Function Family). A prefix puncturable pseudorandom function family scheme (PPRF) consists of the following PPT algorithms.

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

The following properties are required. **Pseudorandomness**: For all PPT adversaries Adv, it holds that:
$$\Pr[\mathsf{Adv}^{\mathsf{F.eval}(s,.)} \text{ outputs } 1\ ] - \Pr[\mathsf{Adv}^{\mathsf{R}(.)} \text{ outputs } 1\ ] \leq \mathsf{negl}(\kappa)$$
where, $s \leftarrow \mathsf{F.gen}(1^\kappa)$ and $\mathsf{R}(.)$ is a random function. **Correctness of Puncturing**: $\forall s, \mathsf{pre}$, it holds that
$$\mathsf{F.eval}(\mathsf{F.punct}(s, \mathsf{pre}), x) = \begin{cases} \mathsf{F.eval}(s, x) \text{ if } x \text{ does not have prefix } \mathsf{pre} \\ \perp \text{ else} \end{cases}$$

Experiment $\textsc{Sig-forge}^{\textsc{punct}}$

Parameter: Let κ be the security parameter.

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

Fig. 6. Puncturable Signature unforgeability security Experiment.

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

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

where $\mathrm{IND}^{\mathrm{PPRF}}$ is shown in Figure 7. ◁

Experiment $\mathbf{IND}^{\mathbf{PPRF}}$

Parameter: Let κ be the security parameter.

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

Fig. 7. Puncturable PRF Security Experiment.

4 The R3PO Framework

4.1 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 π and a deterministic message function μ, both of which can be parameterized by (secret) values α, β (hardwired into circuits $\pi^{(\cdot)}$, $\mu^{(\cdot)}$ respectively).

Definition 13 (Reactive Program over ($\mathcal{X}$, Σ, $\mathcal{A}$, $\mathcal{B}$, M)). A *reactive program* $(\pi^{(\alpha)}, \mu^{(\beta)})$, with input alphabet $\mathcal{X}$, a state-space Σ, a start-state $\mathsf{start} \in \Sigma$ and secret spaces $\mathcal{A}$, $\mathcal{B}$ is specified by a deterministic *transition* program $\pi^{(\alpha)} : \Sigma \times \mathcal{X} \to \Sigma$ parameterized by a secret $\alpha \in \mathcal{A}$ and a deterministic[7] message function $\mu^{(\beta)} : \Sigma \to \mathsf{M}$ parameterized by a secret $\beta \in \mathcal{B}$, that on input sequence $(x_1, \ldots, x_\ell) \in \mathcal{X}$, *reaches* a state σ_ℓ, where $\sigma_i = \pi^{(\alpha)}(\sigma_{i-1}, x_i)$ for $i = 1, \ldots, \ell$ and $\sigma_0 = \mathsf{start}$, and *outputs* a message $\mu^{(\beta)}(\sigma_\ell)$. We also define $\text{REACH}_{\pi^{(\alpha)}}(x_1, \ldots, x_\ell) = \{\sigma_0, \cdots, \sigma_\ell\}$ and $\overline{\pi}^{(\alpha)}(x_1, \ldots, x_\ell) = \sigma_\ell$. $\triangleleft$

Reactive programs have an associated implicit security parameter κ; specifically, we require that the states in Σ and secrets in $\mathcal{A}$, $\mathcal{B}$ are represented as binary strings of length polynomial in the security parameter κ, and the functions $\pi^{(\alpha)}$ and $\mu^{(\beta)}$ are polynomial in κ. Throughout the rest of the paper, we shall omit κ and implicitly refer to "polynomial in κ" 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., $\mathcal{I} : \Sigma \to [N]$ for some positive integer N. We say that $\mathcal{I}$ partitions the state space Σ into $\Sigma_1, \cdots, \Sigma_N$ 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 Σ_1. We say that a transition function $\pi^{(\alpha)}$ is **tree-ordered** with respect to $\mathcal{I}$, if the directed graph over $[N]$ (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.

Definition 14 ($(\mathcal{P},\mathcal{M})$-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

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

to indicate that at the end of the interaction, G outputs $\left((\pi^{(\alpha)}, \mu^{(\beta)}), a_G\right)$ 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. $\triangleleft$

An adversary class $\mathcal{Q}$ is simply a set of adversaries Q. Some useful adversary classes depending on the application are: set of all PPT machines (for active corruption) and set of "semi-honest" PPT machines which follow a given protocol. We also consider adversary classes with *setup*. For any T that is a program in a setup class $\mathcal{T}$, we use Q^T to denote an adversary Q that gets oracle access to an honest execution of T.

4.2 Reach Extractor

To define a reach extractor, we introduce some notation. We write $Q\hat{|}\mathcal{E}$ 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 Q^T in the composite machine $Q^T\hat{|}\mathcal{E}$, $\mathcal{E}$ is allowed to replace T with any program from $\mathcal{T}$. For example, to capture common reference strings as a setup, $\mathcal{T}$

[7] As described in Appendix A.1, restricting our obfuscations to deterministic message functions is without loss of generality, even if we are interested in randomized message functions.

would correspond to $\{\mathsf{Setup}, \mathsf{Setup}_{\mathrm{Sim}}\}$, where Setup is the standard setup algorithm and $\mathsf{Setup}_{\mathrm{Sim}}$ produces a simulated CRS.

$\mathcal{E}$ is a valid reach-extractor if the following hold: in the IDEAL interaction, $\mathcal{E}$ observes the adversary Q and produces an extra output (Π, X^*) such that the states reached in Π 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.[8]

Definition 15 (Reach-Extractor for Q w.r.t. $(\mathcal{G}, \mathring{\mathcal{P}})$). 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 $\mathring{\mathcal{P}}$, 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:

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

output $X \leftarrow D(\pi^{(\alpha)}, \mu^{(\beta)}, a_G, a_Q)$

$\text{IDEAL}(G, Q\hat{|}\mathcal{E}, D)$: $\left((\pi^{(\alpha)}, \mu^{(\beta)}), a_G; a_Q, \Pi, X^*\right) \leftarrow \langle G : Q\hat{|}\mathcal{E} \rangle$

output $X \leftarrow D(\pi^{(\alpha)}, \mu^{(\beta)}, a_G, a_Q)$

and further the following hold:

- In $\text{IDEAL}(G, Q\hat{|}\mathcal{E}, D)$, $\Pi \in \mathring{\mathcal{P}}$.
- Suppose $\mathcal{I}$ partitions Σ into N parts $\Sigma_1, \cdots, \Sigma_N$. Then
  - **Reach-Bound**: For all $i \in [N]$, $\Pr[(\text{REACH}_{\pi^{(\alpha)}}(X) \cap \Sigma_i) \nsubseteq (\text{REACH}_\Pi(X^*) \cap \Sigma_i)]$ is negligible.
  - **Reach-Restriction**: For all $i \in [N]$, $|\text{REACH}_\Pi(X^*) \cap \Sigma_i| \leq 1$. ◁

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 $\Pi \in \mathring{\mathcal{P}}$ to describe the set of states reachable by the adversary Q. While in many of our examples, $\mathring{\mathcal{P}}$ 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 6.1), 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.

4.3 Reach-Restricted Reactive Program Obfuscation

Recall that, our goal in obfuscating a reactive program is to hide the parameters α, β, except for the states an adversary can reach. Let $\mathcal{E}$ be a reach-extractor for Q w.r.t. $\mathcal{G}$ s.t. $\mathcal{E}$ outputs Π, 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 Sim which, given only the circuits $\pi^{(\cdot)}$, $\mu^{(\cdot)}$ and the reachable states $(\overline{x}, \mu^{(\beta)}(\Pi(\overline{x})))$ for input sequences $\overline{x} \in X^*$, can output an obfuscation indistinguishable from a real obfuscation.

Definition 16 (R3PO scheme $\mathcal{O}$ for $(\mathcal{G},\mathcal{Q},\mathring{\mathcal{P}})$). 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 $\mathring{\mathcal{P}}$, if the following hold:[9]

[8] 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).

[9] 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 $\pi^{(\cdot)}$ and $\mu^{(\cdot)}$ (resp.), which take α and β (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 $\overline{x} \in \mathcal{X}^*$, it holds that $\rho(\overline{x}) = \mu^{(\beta)}(\overline{\pi}^{(\alpha)}(\overline{x}))$.

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

$\textsc{real}(G, Q)$:
$\left((\pi^{(\alpha)}, \mu^{(\beta)}), a_G; a_Q\right) \leftarrow \langle G : Q \rangle$
$\rho \leftarrow \mathcal{O}(\pi^{(\alpha)}, \mu^{(\beta)})$
output (ρ, a_G, a_Q)

$\textsc{ideal}(G, Q)$:
$\left((\pi^{(\alpha)}, \mu^{(\beta)}), a_G; a_Q, \Pi, X^*\right) \leftarrow \langle G : Q \hat{|} \mathcal{E} \rangle$
$\rho \leftarrow \mathsf{Sim}\left(\pi^{(\cdot)}, \mu^{(\cdot)}, \Pi, \{\overline{x}, \mu^{(\beta)}(\Pi(\overline{x}))\}_{\overline{x} \in X^*}\right)$
output (ρ, a_G, a_Q)

$\triangleleft$

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 π of a reactive program produced by a generator G, as consisting of several one-step transitions π_i of reactive programs produced by generators H_i. Below, we define the notion of a σ-restriction of π at a state σ.

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

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

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

Note that the state space Σ of π can be exponentially large in κ, and correspondingly π consists of that many one-step transition functions. When decomposing π, we will group them into polynomially many classes of transition functions, using the partition $\mathcal{I}$ of the state space, $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_N$ associated with π. This imposes the following structure on the class of transition functions $\mathcal{P}$ to which π 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 Σ and partition function $\mathcal{I} : \Sigma \to [N]$, we define $\mathcal{P}_1 \times \cdots \times \mathcal{P}_N$ to consist of transition functions $\pi^{(\alpha)}$ such that for each state $\sigma \in \Sigma_i$, the one-step σ-restriction of $\pi^{(\alpha)}$ is in $\mathcal{P}_i$. That is, for all $\sigma \in \Sigma$ and inputs x, $\pi^{(\alpha)}(\sigma, x) = \widehat{\pi}_\sigma^{(\alpha)}(\sigma, x)$ where $\widehat{\pi}_\sigma^{(\alpha)} \in \mathcal{P}_{\mathcal{I}(\sigma)}$.

Though π can have exponentially many states, we would like to view it as composed of N transition functions, $\widehat{\pi}_{\sigma_i} \in \mathcal{P}_i$, 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 π is dynamically generated by a generator G interacting with an adversary Q, and the reachable states in π 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.[10] We also introduce a non-blackbox wrapper W which will be used to adapt an adversary Q (that expects to interact with G) so that it can interact with both G and H.

- For a generator H, we write $\boxed{\boxed{Z}_{H}}$ 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 $\boxed{\boxed{Z}_{H}}$ is bounded by that of H plus an *additive* $\mathsf{poly}(\kappa)$ overhead that depends on Z.
- For a generator G, we write $\boxed{\boxed{G}_{J}}$ to denote a slightly different composite machine, which is similar to $\boxed{\boxed{J}_{G}}$ (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 $\boxed{\boxed{Q}_{W}}$ 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 $\boxed{\boxed{G}_{J}}$, $\boxed{\boxed{Z}_{H}}$ and $\boxed{\boxed{Q}_{W}}$ 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 $\boxed{\boxed{G}_{J}}$ is connected to $\boxed{\boxed{Q}_{W}}$, Q directly interacts with G, whereas when $\boxed{\boxed{Z}_{H}}$ is connected to $\boxed{\boxed{Q}_{W}}$, Q interacts with Z.

For the ease of writing expressions, we shall denote $\boxed{\boxed{G}_{J}}$ by $G[\![J$, and $\boxed{\boxed{Z}_{H}}$ by $H[\![Z$. We will denote $\boxed{\boxed{Q}_{W}}$ by $Q\lfloor W$; in fact, we will be interested in $\boxed{\boxed{Q\hat{\rceil}\mathcal{E}}_{W}}$ (where $Q\hat{\rceil}\mathcal{E}$ itself is a composite machine involving a reach-extractor which interacts with Q as defined in Definition 15); we shall denote it by $Q\hat{\rceil}\mathcal{E}\lfloor W$.

Partial Reach-Extractor: For a valid decomposition, it will be important to have a bi-simulation that maps π to a one-step restriction π_σ such that σ is the unique reachable state in a subset of states Σ_i. To enforce this, we shall rely on an extractor for the adversary Q w.r.t. the generator G that produces π. 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 16). 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 Σ_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 15, but with the relaxation that the reach-bound condition needs to hold only for $i \le t$, instead of $i \le N$. (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 17 (Decomposition of $(\mathcal{G}, \mathcal{Q})$ to $\mathcal{L}$). Let $\mathcal{G}$ be a $(\mathcal{P}, \mathcal{M})$-generator class where $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_N$ is tree-ordered. Let $\mathcal{L} = (\mathcal{H}, \mathcal{Q}, \mathring{\mathcal{P}})$, where $\mathcal{H} = \{H[\![Z \mid \text{PPT } Z\}$ for a fixed $(\mathcal{P}_i, \mathcal{M}_i)$-generator H, $\mathcal{Q}$ is an adversary class, and $\mathring{\mathcal{P}}$ is a transition function class.

[10] 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.

Then, a generator $G \in \mathcal{G}$ is said to be *decomposable at part i to $\mathcal{L}$* 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. $(\mathcal{G}, \mathring{\mathcal{P}})$, it holds that $Q\hat{|}\mathcal{E}|W \in \mathcal{Q}$ and:

- **Indistinguishability:** $\langle G \| J : Q\hat{|}\mathcal{E}\lfloor W \rangle \approx \langle H \| Z : Q\hat{|}\mathcal{E}\lfloor W \rangle$.
- In $\langle G \| J : Q\hat{|}\mathcal{E}\lfloor W \rangle$, let the output of G be $((\pi^{(\alpha)}, \mu^{(\beta)}), a_G)$, and of $\mathcal{E}$ be (a_Q, Π, X^*); then J outputs $((\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_\sigma^{(\widehat{\beta})}), a_J)$ s.t.
  - **Correct One-Step Restriction:** $\text{REACH}_\Pi(X^*) \cap \Sigma_i \subseteq \{\sigma\}$.
  - **Correct Message Function:** $\widehat{\mu}_\sigma^{(\widehat{\beta})} \leftarrow \mathcal{M}_i$ is uniformly sampled at the end of the execution.

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

Above, we require two simulations to produce indistinguishable outputs (which includes their communication, as $Q\hat{|}\mathcal{E}\lfloor W$ outputs its entire view as part of output), with J mimicking H, and Z mimicking G. The "correct one-step restriction" condition forces J (and hence H) to output a one-step restriction whose start state is the state that is reachable, as reported by a (partial) reach-extractor for G.

An Illustrative Example. We illustrate the above decomposition definition via an example. Consider a program corresponding to "Signature + Commitment" that 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 6. We consider a generator G which samples a signature key-pair $(\mathsf{sk}, \mathsf{vk})$, 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 $\mathcal{G}^{\cdot}_{\text{COMMIT}}$ 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 $(\mathsf{sk}, \mathsf{vk})$ 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.[11]

It can be seen that J, Z, W as 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 σ 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 $\widehat{\mu}_i$ of these one-step programs as well.

As described in Section 2.3, $\widehat{\mu}_i$ 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 $\widehat{\beta}$) which takes the part index i as an input, along with a bit position j and bit value b. Then $\widehat{\mu}_i$ will be of the form $\mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},t}$ defined below, which only retains the part of $\widehat{\beta}$ for parts $i > t$.

[11] 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 $\mathcal{P}_i \times \widehat{\mathcal{M}}_{\mathcal{I},i}$)

Garbling Scheme: Let $(\mathsf{GCGarble}, \mathsf{GCEval})$ 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 $\widehat{\beta} : [N] \times [n] \times \{0,1\} \to \{0,1\}^\kappa$
- For each $i \in [N]$,
  - Define $\widehat{\mu}_i$ to be $\mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},i}$.
  - Define the function f_i as follows (with a fresh random tape hard-coded for $\mathcal{O}_i$ and, if needed, $\mu^{(\beta)}$):

$$f_i(\sigma) \;=\; \left(\mathcal{O}_i(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_i),\ \mu^{(\beta)}(\sigma)\right)$$

  - Let $\mathrm{GC}_i \leftarrow \mathsf{GCGarble}(f_i, \widehat{\beta}_i)$, where $\widehat{\beta}_i(\ell, b) = \widehat{\beta}(i, \ell, b)$ for $\ell \in [n], b \in \{0,1\}$.
- Let $i_0 = \mathcal{I}(\mathsf{start})$, where start is the start-state of $\pi^{(\alpha)}$.
  Output $\left(f_{i_0}(\mathsf{start}), \{\mathrm{GC}_i\}_{i \in [N] \setminus \{i_0\}}\right)$, along with a "driver program."

Fig. 8. Obfuscator $\mathcal{O}$ used to prove Theorem 1.

Definition 18 (Message function space $\widehat{\mathcal{M}}$). Let $\Sigma = \{0,1\}^n$, with a partition function $\mathcal{I} : \Sigma \to [N]$, and $\widehat{\beta} : [N] \times [n] \times \{0,1\} \to \{0,1\}^\kappa$. A *state labeling function* $\mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},t} : \Sigma \to \{0,1\}^{n\kappa}$ is defined as

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

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

$$\widehat{\mathcal{M}}_{\mathcal{I},t} \;=\; \left\{\left(\mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},t}\right) \mid \widehat{\beta} : [N] \times [n] \times \{0,1\} \to \{0,1\}^\kappa\right\} \qquad \triangleleft$$

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

Proof Sketch. The obfuscator $\mathcal{O}$ is shown in Figure 8. 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 $G \in \mathcal{G}$ and any $Q \in \mathcal{Q}$. For each $i \in \{1, 2, \cdots, N\}$, we define a sequence of hybrid experiments $\mathsf{Hybrid}^{i,0}$ through $\mathsf{Hybrid}^{i,6}$, and argue indistinguishability of the outputs from adjacent hybrids, and further show that the outcome of $\mathsf{Hybrid}^{i,6}$ is indistinguishable from that of $\mathsf{Hybrid}^{i+1,0}$. Finally, the outcome of the real obfuscation experiment will be shown to be identical to that of $\mathsf{Hybrid}^{1,0}$ and that of the ideal obfuscation experiment will be shown to be identical to that of $\mathsf{Hybrid}^{N+1,0}$.

For each $i \in [N]$, as we go from $\mathsf{Hybrid}^{i,0}$ to $\mathsf{Hybrid}^{i+1,0}$, 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 $\mathcal{E}_i$ corresponding to $\mathcal{H}_i$ w.r.t. Q. This is immediate from the tree-ordering property of reactive programs in $\mathcal{P}$[12]. The corresponding partial simulation will simulate the garbled circuit for partition i on a simulated obfuscation of a one-step σ_i-restricted program, where $\{\sigma_i\}$ is as extracted by E_{i-1} and bounds the reach of the adversay in partition i.

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

– $\mathsf{Hybrid}^{i,1}$: In this hybrid, the i^{th} garbled circuit is simulated on an input 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 σ_i). Indistinguishability follows from a combination of reach restriction of E_{i-1} and garbled circuit security.

– $\mathsf{Hybrid}^{i,2}$: 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 $Q\hat{|}E_{i-1}\lfloor W_i$. J_i outputs a one-step σ_i-restricted reactive program, where σ_i is the state in partition i reachable by E_{i-1}. Indistinguishability holds since, the interaction between G and $Q\hat{|}E_{i-1}$ is unchanged, and so is the rest of the experiment.

– $\mathsf{Hybrid}^{i,3}$: In this hybrid, we use the decomposition of G to $\mathcal{L}_i$ to have $H_i\|Z_i$ interact with $Q\hat{|}E_{i-1}\lfloor W_i$.

– $\mathsf{Hybrid}^{i,4}$: In this hybrid, we use the one-step extractor $\mathcal{E}_i$ and obfuscation simulator Sim_i for H_i from the library to simulate the $f_i(\sigma_i)$ (obfuscation of the one-step program at σ_i). This finishes the simulation of the i^{th} garbled circuit. Indistinguishability follows from R3PO security of $\mathcal{O}_i$.

– $\mathsf{Hybrid}^{i,5}$: In this hybrid, we rewrite the above as an interaction between $G\|J_i$ and $Q\hat{|}E_{i-1}\lfloor W_i\hat{|}\mathcal{E}_i$. Indistinguishability follows from the property of W_i, that it outputs its entire state as auxiliary output. Thus, decomposition holds even if a passive program $\mathcal{E}_i$ is allowed to see inside W_i.

– $\mathsf{Hybrid}^{i,6}$: 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 $E_{i-1}\lfloor W_i\hat{|}\mathcal{E}_i$ and J_i). Indistinguishability with $\mathsf{Hybrid}^{i,5}$ follows from the properties of J_i. Also hybrid $\mathsf{Hybrid}^{i,6}$ and $\mathsf{Hybrid}^{i+1,0}$ are identical to each other.

Please refer to Appendix A for the full proof that it is an R3PO scheme.

6 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).

6.1 Commitment Opening R3PO

Let $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ 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 hardwired in the program (and fixed during the interaction).

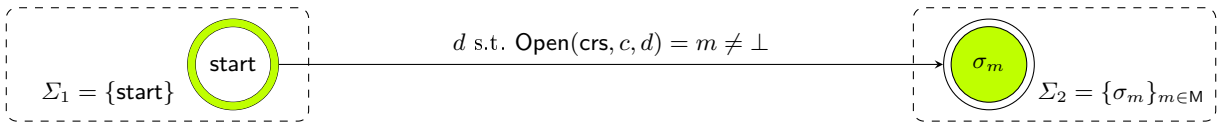

Fig. 9. Transition function $\pi_{c,\mathsf{crs}}$ is hardwired with a commitment c and a setup crs. On input d, it transitions from start to $\sigma_m \in \Sigma_2$ iff d is a valid opening of c to m.

[12] 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 $\mathcal{E}_k$

Generator class and Program family. For a commitment scheme Com and message function class $\mathcal{M}$, the generator class $\mathcal{G}^{\mathsf{Com},\mathcal{M}}_{\text{COMMIT}}$ is defined as a set of generators of the form $H\|Z$ for a fixed H and all polynomial Z [13]. The generator H behaves as follows. It accepts a setup 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 $(\pi_{c,\mathsf{crs}}, \mu^{(\beta)})$, whose transition function w.r.t. a state space $\Sigma = \{\mathsf{start}\} \cup \{\sigma_m | m \in \mathsf{M}\}$ is specified as follows:

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

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

Adversary class and Reach-Extractor. Let $\mathsf{Setup}_{\text{Sim}}$ denote the setup algorithm used by a *simulator* for the security experiment of Com; then the adversary class $\mathcal{Q}^{\mathsf{Com}}_{\text{COMMIT}}$ is defined to have adversaries of the form Q^T, where $T \in \{\mathsf{Setup}, \mathsf{Setup}_{\text{Sim}}\}$ and Q is an arbitrary PPT program that runs T honestly, receives crs from it, fixes a commitment c arbitrarily and sends (crs, c) to the generator.

In the real experiment of Definition 16, T will correspond to Setup (the setup algorithm of Com). In the ideal experiment, we define an extractor $\mathcal{E}$ as follows. It replaces T to be the simulated setup $\mathsf{Setup}_{\text{Sim}}$, which outputs crs and a trapdoor for crs. When Q outputs a commitment c, it runs the extractor of Com (using the trapdoor of crs) to extract the message m from c. However, recall that the extractor of the commitment scheme cannot directly extract the commitment opening d, and thus $\mathcal{E}$ cannot output a valid input for the real transition function. Instead, we will have $\mathcal{E}$ output $X^* = \{m\}$ and a relaxed transition function $\Pi_m \in \mathring{\mathcal{P}}_{\text{COMMIT}}$ defined as: $\Pi_m(start, m') = \sigma_m$ if $m = m'$. It is easy to verify that $\mathcal{E}$ is a valid reach-extractor.

R3PO. In Appendix B.1 we show how to construct a commitment scheme Com and an R3PO w.r.t. $(\mathcal{G}^{\mathsf{Com},\mathcal{M}}_{\text{COMMIT}}, \mathcal{Q}^{\mathsf{Com}}_{\text{COMMIT}}, \mathring{\mathcal{P}}_{\text{COMMIT}})$ from any UC-secure 2-round OT. This construction repurposes the OT scheme as a commitment scheme, following [7, 30]. 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]). If the OT scheme is semi-honest secure, then we instead get a weakly-secure commitment scheme (Definition 4). We state the result as the following lemma.

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

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

6.2 Signature Checking

Let $\mathsf{Sig} = (\mathsf{gen}, \mathsf{sign}, \mathsf{verify}, \mathsf{punct}, \mathsf{psign})$ be a puncturable signature scheme (Definition 11). 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 vk, and transitions from the start state to the state σ_m on input a signature τ if vk verifies (m, τ). All states σ_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 vk. We discuss this in Section 6.2.1 for the general case: Q specifies a target message prefix mpre.
- A semi-honest adversary Q specifies the verification key vk. We discuss this in Section 6.2.2.

[13] the generator $H\|Z$ is designed so that, Z's only role is to specify the message function $\mu^{(\beta)}$

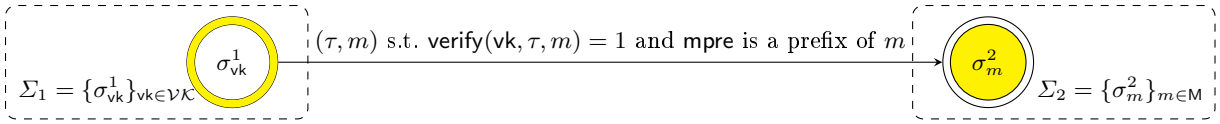

Fig. 10. Transition function $\pi_{\mathsf{mpre}}[\sigma^1_{\mathsf{vk}}]$ has mpre hardwired in it and vk encoded in the start state $\sigma^1_{\mathsf{vk}} \in \Sigma_1$. On input (τ, m), it transitions to σ^2_m iff τ is a valid signature on m w.r.t. vk and mpre is a prefix of m.

6.2.1 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.

Generator class and Program Family. The generator class $\mathcal{G}^{\mathsf{Sig},\mathcal{M}}_{\text{SIGN}}$ is a set of generators of the form $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}\|Z$ for a fixed $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}$ and all polynomial time Z. The generator $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}$ behaves as follows. It samples a signature key pair (sk, vk) and sends vk to Q. It then accepts polynomial queries $m_i \in \mathsf{M}$ from Q and responds with $\mathsf{sign}(sk, m)$. It then accepts a target message m from Q, sets mpre to be the t-bit prefix of m, verifies that at most a single signature with mpre was given out and punctures the key as $\mathsf{punct}(\mathsf{sk}, \mathsf{mpre})$. It finally accepts a message function $\mu^{(\beta)} \in \mathcal{M}$ from Z and outputs a reactive program $(\pi_{\mathsf{mpre}}[\sigma^1_{vk}], \mu^{(\beta)}, s)$ with start state σ^1_{vk}, whose transition function w.r.t. a state space $\Sigma = \{\sigma^1_{vk} | vk \in \mathcal{VK}\} \cup \{\sigma^2_m | m \in \mathsf{M}\}$ is specified as follows:

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

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

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 τ. There is a trivial extractor that simply outputs $\pi_{\mathsf{mpre}}[\sigma^1_{\mathsf{vk}}]$ and $X^* = \{(m, \tau)\}$ from the transcript of Q's interaction.

R3PO. In Appendix B.2.3, we show how to construct a puncturable signature scheme Sig and an R3PO w.r.t. $(\mathcal{G}^{\mathsf{Sig},\mathcal{M}}_{\text{SIGN}}, \mathcal{Q}^{\mathsf{Sig}}_{\text{SIGN}}, \mathcal{P}^{\mathsf{Sig}}_{\text{SIGN}})$ from any one-time signature with encryption (OTSE) scheme, which in turn can be instantiated from the Decisional Diffie-Hellman assumption (DDH). Our construction borrows from results and ideas in [21, 22]. We state the result as the following lemma.

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

6.2.2 R3PO for Signature-Checking with Adversarial key

In this section, we consider the case where adversary specifies the signature key. In this case, note that the generalization to fix only the prefix mpre in the reactive program is not reach-restricted, 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 σ_m on input a message m and signature τ if vk verifies (τ, m).

Generator class and Program Family. The generator class is a set of generators of the form $H\|Z$ 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 σ^1_{vk}, whose transition function w.r.t. a state space $\Sigma = \{\sigma^1_{vk} | vk \in \mathcal{VK}\} \cup \{\sigma^2_m | m \in \mathsf{M}\}$ is specified as follows:

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

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

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

R3PO. There exists a trivial obfuscation scheme for $\mathcal{G}^{\mathsf{Sig}}_{\text{QSIGN}}$ via a general epsilon transition (refer Section 6.4). Nevertheless, we show that the obfuscation scheme described in Appendix B.2.3 for signature checking with generator key, is infact also a secure obfuscation scheme for signature checking with adversary key. We state the result as the following lemma.

Lemma 3. *If there exists a semi-honest secure OTSE scheme, then there exists a puncturable signature scheme* Sig *and a PPT program* $\mathcal{O}_{\mathsf{Sig}}$ *s.t.* $\mathcal{O}_{\mathsf{Sig}}$ *is an R3PO obfuscation scheme w.r.t.* $(\mathcal{G}^{\mathsf{Sig}}_{QSIGN}, \mathcal{Q}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{QSIGN})$ *as well as* $(\mathcal{G}^{\mathsf{Sig}}_{SIGN\text{-}PO}, \mathcal{Q}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{SIGN\text{-}PO})$.

6.3 Hash with Selective Opening

Let $\mathsf{Hash} = (\mathsf{crsGen}, \mathsf{hash}, \mathsf{openBlock}, \mathsf{acceptBlock})$ be a block-openable collision-resistant hash function scheme (Definition 6), wherein the hash digest on a matrix D can be opened via the $\mathsf{openBlock}$ 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.

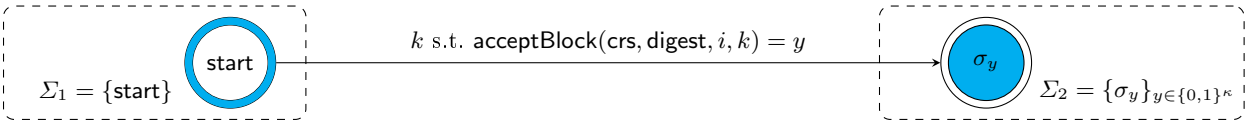

Fig. 11. Transition Program $\pi_{\mathsf{digest},i,\mathsf{crs}}$ for Hash with Selective Opening.

Generator class and Program Family. The generator class $\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}$ 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 generates $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa)$, sends it to Q, and then reads a hash digest digest and a block index i from Q. It also accepts a message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}$ from Z. It then outputs a reactive program $(\pi_{\mathsf{digest},i,\mathsf{crs}}, \widehat{\mu}^{(\widehat{\beta})})$, whose transition function w.r.t. a state space $\Sigma = \{\mathsf{start}\} \cup \{\sigma_y | y \in \mathsf{M}\}$ is specified as follows:

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

The partition function is defined as: $\mathcal{I}_{\text{HASH}}(\mathsf{start}) = 1$ and $\mathcal{I}_{\text{HASH}}(\sigma_y) = 2$ for all $y \in \mathsf{M}$.

Adversary class and Reach-Extractor. The adversary class is defined as the set of all PPT Q^T, where T is a setup in $\{\mathsf{crsGen}\}$ that samples the crs and each Q honestly constructs the hash of a message $D \in \mathsf{M}$ as $\mathsf{digest} = \mathsf{hash}(\mathsf{crs}, y)$. 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 $\pi_{\mathsf{digest},i,\mathsf{crs}}$, $X^* = \{k\}$.

R3PO. In Appendix B.3 we show how to construct an arbitrarily-compressing hash scheme Hash and an R3PO scheme w.r.t. $(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}, \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}}, \mathring{\mathcal{P}}^{\mathsf{Hash}}_{\text{HASH}})$, assuming a laconic OT scheme with factor-2 compression. We state the result as the following lemma.

Lemma 4. *If there exists a laconic OT scheme with factor-2 compression, then there exists an arbitrarily-compressing hash scheme* Hash *and an R3PO scheme w.r.t.* $(\mathcal{G}^{\mathsf{Hash}}_{HASH}, \mathcal{Q}^{\mathsf{Hash}}_{HASH}, \mathring{\mathcal{P}}^{\mathsf{Hash}}_{HASH})$.

6.4 Epsilon-Transition R3PO

The generator class $\mathcal{G}^{\Sigma}_{\epsilon}$ 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 σ_1 (as the start state) to σ_2 on input ϵ. 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 σ_1 to $f(\sigma_1) \in \Sigma_2$ on input ϵ. The corresponding partition function is simply $\mathcal{I}_\epsilon(\sigma) = i$ if $\sigma \in \Sigma_i$.

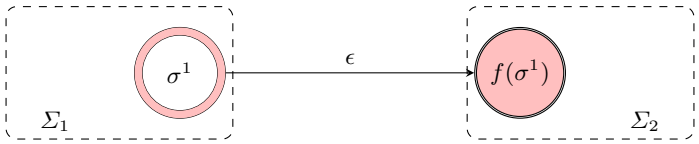

Fig. 12. Transition Program π_f

An R3PO obfuscation scheme for Epsilon Transition. There is a trivial R3PO obfuscation scheme for $\mathcal{P}_\epsilon^\Sigma$ w.r.t. adversary class $\mathcal{Q}$ (all PPT Q) and a relaxed program class same as $\mathcal{P}_\epsilon^\Sigma$):

$$\mathcal{O}_\epsilon(\pi_f, \widehat{\mu}^{(\widehat{\beta})}) = \widehat{\mu}^{(\widehat{\beta})}(f(\sigma^1))$$

6.5 Generalizing the State Space and Partition Function

Let $\mathcal{G}$ be a $(\mathscr{P}, \widehat{\mathcal{M}}, \mathcal{I})$-generator class of the form $H \| Z$, for a fixed H and PPT Z (where $\mathscr{P}$ is defined w.r.t. a state space Σ and $\mathcal{I} : |\Sigma| \to [N]$ is a partition function), and $\mathcal{O}$ be an R3PO obfuscation for $(\mathcal{G}, \mathcal{Q}, \mathring{\mathscr{P}})$. Let P be a set of prefixes, $\Sigma' = \mathsf{P} \,||\, \Sigma$[14] be a state space, and $\mathcal{I}' : |\Sigma'| \to [N']$ be a partition function s.t. there exists an injective map $\eta : [N] \to [N']$ which satisfies

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

We define the following wrapper programs. In particular, for all $\mathsf{pre} \in \mathsf{P}$, $\pi^{(\alpha)} \in \mathscr{P}$, we describe the program $\mathsf{WrapOne}(\mathsf{pre}, \pi^{(\alpha)})$ in Figure 13.

Input: (σ', x).

– Set y as

$$y = \begin{cases} \mathsf{pre} \,||\, \pi^{(\alpha)}(\sigma, x) & \text{if } \exists \sigma \text{ s.t. } \sigma' = \mathsf{pre}\|\sigma, \\ \sigma' & \text{otherwise.} \end{cases}$$

– Output y.

Fig. 13. Wrapper Program $\mathsf{WrapOne}(\mathsf{pre}, \pi^{(\alpha)})$

Similarly, for all $\mathsf{pre} \in \mathsf{P}$, $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}$, we define the program $\mathsf{WrapTwo}(\mathsf{pre}, \widehat{\mu}^{(\widehat{\beta})}, \widehat{\mu}^{(\widehat{\beta})'})$ in Figure 14.

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

– Program family $\mathscr{P}'$: Every program in $\mathscr{P}'$ is defined as a wrapper on a corresponding program $\pi^{(\alpha)} \in \mathscr{P}$. That is,

$$\mathscr{P}' = \{\mathsf{WrapOne}(\mathsf{pre}, \pi^{(\alpha)}) \mid \mathsf{pre} \in \mathsf{P}, \pi^{(\alpha)} \in \mathscr{P}\}$$

– Program family $\mathring{\mathscr{P}}'$: This program family is same as $\mathscr{P}'$.

[14] We basically consider all states of the form $\{\mathsf{pre}\|\sigma\}_{\mathsf{pre}\in\mathsf{P}, \sigma\in\Sigma}$.

Input: σ'.

- Parse $\widehat{\beta}'$ as $(\widehat{\beta}_0, \widehat{\beta}_1)$.
- If $\widehat{\mu}^{(\widehat{\beta})} \neq \mathsf{encode}_{\widehat{\beta}_1}(.)$ then abort.
- Else, set y as

$$y = \begin{cases} \left(\mathsf{encode}_{\widehat{\beta}_0}(\mathsf{pre}),\ \mathsf{encode}_{\widehat{\beta}_1}(\sigma)\right) & \text{if } \exists \sigma \text{ s.t. } \sigma' = \mathsf{pre}\|\sigma, \\ \perp & \text{otherwise.} \end{cases}$$

- Output y.

Fig. 14. Wrapper Program $\mathsf{WrapTwo}(\mathsf{pre}, \widehat{\mu}^{(\widehat{\beta})}, \widehat{\mu}^{(\widehat{\beta})'})$

- A $(\mathscr{P}', \widehat{\mathcal{M}})$-Generator Class $\mathcal{G}'$: This class contains all generators of the form $H' \| Z'$ for all PPT Z' and a fixed H', where H' behaves as follows: it internally runs H, allows H to interact with Z', gets output $((\pi^{(\alpha)}, \mu^{(\beta)}), a_G)$ from H, receives $(\mathsf{pre}, \widehat{\mu}^{(\widehat{\beta})'}, \mathcal{I}')$ from Z' and finally outputs $((\pi^{(\alpha)'}, \widehat{\mu}^{(\widehat{\beta})'}), a_G)$, where $\pi^{(\alpha)'} = \mathsf{WrapOne}(\mathsf{pre}, \pi^{(\alpha)})$ and $\widehat{\mu}^{(\widehat{\beta})'} = \mathsf{WrapTwo}(\mathsf{pre}, \widehat{\mu}^{(\widehat{\beta})}, \widehat{\mu}^{(\widehat{\beta})'})$.

We now build an obfuscator scheme for $(\mathcal{G}', \mathcal{Q}, \mathring{\mathscr{P}}')$ (refer Figure 15).

Preliminaries: Let $\mathcal{G}$ be a $(\mathscr{P}, \widehat{\mathcal{M}}, \mathcal{I})$-generator class and $\mathcal{O}$ be an R3PO obfuscation for $(\mathcal{G}, \mathcal{Q}, \mathring{\mathscr{P}})$. Also, let pre be a fixed state prefix, $\mathcal{I}'$ be a partition function and η be an injective map s.t. $\forall \sigma \in \Sigma, \eta(\mathcal{I}(\sigma)) = \mathcal{I}'(\mathsf{pre}||\sigma)$.

Input: $(\pi^{(\alpha)'}, \widehat{\mu}^{(\widehat{\beta})'})$.

- Parse $\pi^{(\alpha)'}$ as $\mathsf{WrapOne}(\mathsf{pre}, \pi^{(\alpha)})$.
- Parse $\widehat{\beta}'$ as $(\widehat{\beta}_0, \widehat{\beta}_1)$ and $\widehat{\mu}^{(\widehat{\beta})'}$ as $\mathsf{WrapTwo}(\mathsf{pre}, \widehat{\mu}^{(\widehat{\beta})}, \widehat{\mu}^{(\widehat{\beta})'})$.
- Construct $\rho \leftarrow \mathcal{O}((\pi^{(\alpha)}, \mu^{(\beta)}))$.
- Output $\left(\mathsf{encode}_{\widehat{\beta}_0}(\mathsf{pre}),\ \rho\right)$.

Fig. 15. Obfuscator $\mathcal{O}'$ for $(\mathcal{G}', \mathcal{Q}, \mathring{\mathscr{P}}')$

Lemma 5. *Let $\mathcal{G}$ be a $(\mathscr{P}, \widehat{\mathcal{M}}, \mathcal{I})$-generator class and $\mathcal{O}$ be an R3PO obfuscation for $(\mathcal{G}, \mathcal{Q}, \mathring{\mathscr{P}})$. Then for all prefixes $\mathsf{pre} \in \mathsf{P}$, partition function $\mathcal{I}'$ and injective map η s.t. $\forall \sigma \in \Sigma, \eta(\mathcal{I}(\sigma)) = \mathcal{I}'(\mathsf{pre}||\sigma)$, there exists an R3PO scheme for $(\mathcal{G}', \mathcal{Q}, \mathring{\mathscr{P}}')$.*

Proof: To show that the scheme given in Figure 15 is a valid obfuscation scheme for $(\mathcal{G}', \mathcal{Q}, \mathring{\mathscr{P}}')$, we need to define a simulator Sim' and an extractor $\mathcal{E}'$ satisfying our R3PO definition. Since $\mathcal{O}$ is a valid obfuscation scheme for $(\mathcal{G}, \mathcal{Q}, \mathring{\mathscr{P}})$, there must exist a simulator Sim and extractor $\mathcal{E}$ for security. $\mathcal{E}'$ can simply be defined to use $\mathcal{E}$ and Sim' can also be defined to use 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. Plus 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. □

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 $\mathbb{A}_1, \ldots, \mathbb{A}_{\mathsf{N}}$, s.t. each authority $\mathbb{A}_i$ publishes its public key mpk_i after a local non-interactive setup. Each authority $\mathbb{A}_i$ also has an attribute-granting policy Θ_i^{gid} w.r.t. each gid. A sender encrypts a message m with a ciphertext-policy ϕ under the public keys of all the authorities, s.t. a receiver with global identifier gid and attribute vector $\bar{\mathsf{x}}$ can decrypt it only if it has attribute key for $\bar{\mathsf{x}}$ and each authorities' attribute-granting policies accepts (that is, $\forall i \in [N], \Theta_i^{\mathsf{gid}}(\bar{\mathsf{x}}) = 1$) and the ciphertext-policy accepts (that is, $\phi(\bar{\mathsf{x}}) = 1$). To get the attribute key for attribute vector $\bar{\mathsf{x}}$, the receiver sends a key-request req_i to each authority $\mathbb{A}_i$, gets back a key-component $\mathsf{sk}_{\mathsf{req}_i}$, and combines all the key-components to construct the key $\mathsf{sk}_{\bar{\mathsf{x}}}$.

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 $\bar{\mathsf{x}}$ 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 ϕ, if it did not send a key-request for any $\bar{\mathsf{x}}$ that satisfies ϕ to the honest authorities.

Definition 19 (Private Multi-Authority ABE (p-MA-ABE)). A p-MA-ABE scheme for N authorities, message space M, class $\mathcal{C}$ of ciphertext-policies and class $\boldsymbol{\Theta}$ of attribute-granting policies, both over n-bit attributes, and global identifiers space GID consists of PPT algorithms as follows:

- $\mathsf{SetupAuth}(1^\kappa) \to (\mathsf{mpk}, \mathsf{msk})$: On input the security parameter κ, outputs the master keys for an individual authority.
- $\mathsf{Encrypt}\left(\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \phi, m\right) \to \mathsf{ct}$: On input the master public keys of all authorities, a policy $\phi : \{0,1\}^n \to \{0,1\}$ in $\mathcal{C}$ and a message $m \in \mathsf{M}$, outputs a ciphertext ct.
- $\mathsf{KeyRequest}\left(\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \mathsf{gid}, \bar{\mathsf{x}}\right) \to (\mathsf{st}, \{\mathsf{req}_i\}_{i\in[\mathsf{N}]})$: On input the master public keys of all authorities, a global identity $\mathsf{gid} \in \mathsf{GID}$ and an attribute vector $\bar{\mathsf{x}} \in \{0,1\}^n$, outputs a recipient state st and requests $\{\mathsf{req}_1, \ldots, \mathsf{req}_{\mathsf{N}}\}$.
- $\mathsf{KeyGen}\left(i, \mathsf{msk}_i, \Theta_i^{\mathsf{gid}}, \mathsf{req}_i, \{\mathsf{mpk}_j\}_{j\in[\mathsf{N}]}\right) \to \mathsf{sk}_{\mathsf{req}_i}$: On input an authority index $i \in [\mathsf{N}]$, master secret key msk_i, an attribute-granting policy Θ_i^{gid}, a request req_i and the master public keys of all authorities, outputs a key component $\mathsf{sk}_{\mathsf{req}_i}$ or $\perp$.
- $\mathsf{KeyCombine}\left(\mathsf{st}, \{\mathsf{sk}_{\mathsf{req}_i}\}_{i\in[\mathsf{N}]}\right) \to \mathsf{sk}_{\bar{\mathsf{x}}}$: On input a receipient state st and a set of key components $\{\mathsf{sk}_{\mathsf{req}_i}\}_{i\in[\mathsf{N}]}$, outputs a secret key $\mathsf{sk}_{\bar{\mathsf{x}}}$.
- $\mathsf{Decrypt}\left(\mathsf{sk}_{\bar{\mathsf{x}}}, \mathsf{ct}\right) \to m$: On input a secret key $\mathsf{sk}_{\bar{\mathsf{x}}}$ and a ciphertext ct, outputs a message m or $\perp$.

The following correctness and security properties are required:

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

$$
\begin{aligned}
\forall i \in [\mathsf{N}], \quad (\mathsf{mpk}_i, \mathsf{msk}_i) &\leftarrow \mathsf{SetupAuth}(1^\kappa) \\
(\mathsf{st}, \{\mathsf{req}_i\}_{i\in[\mathsf{N}]}) &\leftarrow \mathsf{KeyRequest}(\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \mathsf{gid}, \bar{\mathsf{x}}) \\
\forall i \in [\mathsf{N}], \quad \mathsf{sk}_{\mathsf{req}_i} &\leftarrow \mathsf{KeyGen}\left(i, \mathsf{msk}_i, \Theta_i^{\mathsf{gid}}, \mathsf{req}_i, \{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}\right) \\
\mathsf{sk}_{\bar{\mathsf{x}}} &\leftarrow \mathsf{KeyCombine}\left(\mathsf{st}, \{\mathsf{sk}_{\mathsf{req}_i}\}_{i\in[\mathsf{N}]}\right)
\end{aligned}
$$

then $\Pr\left[\mathsf{Decrypt}\left(\mathsf{sk}_{\bar{\mathsf{x}}}, \mathsf{Encrypt}(\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \phi, m)\right) = m\right] = 1$.

2. **Security of encryption**: For any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ with semi-honest corruption of any subset of authorities and malicious corruption of the receiver, there exists a negligible function $\mathsf{negl}(.)$ such that the following holds in the experiment $\text{IND}^{\text{p-MA-ABE}}_{\text{mesg}}$ shown in Figure 16:

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

Parameter: Let κ be the security parameter, N be the number of authorities.

Experiment $\text{IND}^{\text{p-MA-ABE}}_{\text{mesg}}$

- $(\mathsf{st}_0, H) \leftarrow \mathcal{A}_0(1^\kappa)$.
- $\{(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{SetupAuth}(1^\kappa)\}_{i \in H}$.
- $\left(\mathsf{st}_1, \{\mathsf{mpk}_i\}_{i \in [\mathsf{N}] \setminus H}\right) \leftarrow \mathcal{A}_1(\mathsf{st}_0, \{\mathsf{mpk}_i\}_{i \in H})$.
  Let $O_k(y, \{\Theta_i^{\mathsf{gid}}\}_{i \in H}) := \mathsf{KeyGen}\left(k, \mathsf{msk}_k, \Theta_k^{\mathsf{gid}}, y, \{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}\right)$, where $y = (\mathsf{gid}, c)$.
- $(\mathsf{st}_2, \phi, m_0, m_1) \leftarrow \mathcal{A}_2^{\{O_k(.,.)\}_{k \in H}}(\mathsf{st}_1)$
- $b \leftarrow \{0,1\}$.
- $\mathsf{ct} \leftarrow \mathsf{Encrypt}\left(\{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}, \phi, m_b\right)$.
- $b' \leftarrow \mathcal{A}_3^{\{O_k(.)\}_{k \in H}}(\mathsf{st}_2, \mathsf{ct})$.
- If $\mathcal{A}_2$ or $\mathcal{A}_3$ queried any oracle with different key-policies $\{\Theta_i^{\mathsf{gid}}\}_{i \in H}$ for the same globalid gid, output $a \leftarrow \{0,1\}$.
  If $\exists \mathsf{gid}, \bar{\mathsf{x}}$ s.t. $\phi(\bar{\mathsf{x}}) = 1$ and, $\forall k \in H$, $\Theta_k^{\mathsf{gid}}(\bar{\mathsf{x}}) = 1$, and $\exists r, k^* \in H$ s.t. $\mathcal{A}_2$ or $\mathcal{A}_3$ queried the oracle O_{k^*} with req_{k^*}, where $(\mathsf{st}, \{\mathsf{req}_k\}_{k \in [\mathsf{N}]}) \leftarrow \mathsf{KeyRequest}(\{\mathsf{mpk}_k\}_{k \in [\mathsf{N}]}, \mathsf{gid}, \bar{\mathsf{x}};\ r)$, then output $a \leftarrow \{0,1\}$.
  Else, output $a = b \oplus b'$.

Experiment $\text{IND}^{\text{p-MA-ABE}}_{\text{attr}}$

- $(\mathsf{st}_0, \{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}, \bar{\mathsf{x}}_0, \bar{\mathsf{x}}_1, \mathsf{gid}) \leftarrow \mathcal{A}_0(1^\kappa)$.
- $b \leftarrow \{0,1\}$.
- $b' \leftarrow \mathcal{A}_1(\mathsf{st}_0, \mathsf{KeyRequest}(\{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}, \mathsf{gid}, \bar{\mathsf{x}}_b))$.
- Output $b \oplus b'$.

Fig. 16. p-MA-ABE security experiments.

3. **Receiver Privacy against Semi-Honest Adversary**: For any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ that corrupts each authority in a semi-honest way, there exists a negligible function $\mathsf{negl}(.)$ such that the following holds in the experiment $\text{IND}^{\text{p-MA-ABE}}_{\text{attr}}$ shown in Figure 16:

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

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 2)
- a non-interactive UC secure commitment scheme (Definition 3)
- a puncturable signature scheme (Definition 11)
- an R3PO scheme $\mathcal{O}_{\text{p-MA-ABE}}$ w.r.t. $(\mathcal{G}^1_{\text{p-MA-ABE}}, \mathcal{Q}_{\text{p-MA-ABE}})$ and $(\mathcal{G}^2_{\text{p-MA-ABE}}, \mathcal{Q}_{\text{p-MA-ABE}})$, which we describe in Appendix C.2.

Let the number of authorities be N, space of access policies $\mathcal{C}$ 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 $\mathsf{Com.crs}$, for any c, d s.t. $\mathsf{Com.Open}(\mathsf{Com.crs}, c, d) = m$, it holds that (m, c, d) lies in the support of the commit algorithm, that is: there exists r s.t. $(c, d) \leftarrow \mathsf{Com.Commit}(\mathsf{Com.crs}, m; r)$. Any commitment scheme can be enhanced to have this property, by modifying it as follows.

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

Protocol for Private Multi-Authority ABE

Parameter: Let κ be the security parameter, N be the number of authorities.
Primitives:
A CP-ABE scheme $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Encrypt}, \mathsf{ABE.Decrypt})$
A Commitment scheme $\mathsf{Com} = (\mathsf{Com.Setup}, \mathsf{Com.Commit}, \mathsf{Com.Open})$
A puncturable signature scheme $\mathsf{Sig} = (\mathsf{Sig.gen}, \mathsf{Sig.sign}, \mathsf{Sig.verify})$
An obfuscation scheme $\mathcal{O}_{\text{p-MA-ABE}}$

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

– p-MA-ABE.$\mathsf{Encrypt}(\mathsf{pk}, \phi, m)$:
- For all $i \in [\mathsf{N}]$, parse mpk_i as $(\mathsf{Com.crs}_i, \mathsf{Sig.vk}_i, \mathsf{ABE.mpk}_i)$.
- Sample $s_1, \ldots, s_\mathsf{N}$ s.t. $s_1 + \ldots + s_\mathsf{N} = m$.
- For all $i \in [\mathsf{N}]$, compute $\mathsf{ABE.ct}_i \leftarrow \mathsf{ABE.Encrypt}(\mathsf{ABE.mpk}_i, \phi, s_i)$.
- Output $\mathsf{ct} := \{\mathsf{ABE.ct}_1, \ldots, \mathsf{ABE.ct}_\mathsf{N}\}$.

– p-MA-ABE.$\mathsf{Decrypt}(\mathsf{sk}_{\bar{\mathsf{x}}}, \mathsf{ct})$:
- Parse $\mathsf{sk}_{\bar{\mathsf{x}}}$ as $\{\mathsf{sk}^{\mathsf{ABE}}_{\bar{\mathsf{x}},1}, \ldots, \mathsf{sk}^{\mathsf{ABE}}_{\bar{\mathsf{x}},\mathsf{N}}\}$.
- Parse ct as $\{\mathsf{ABE.ct}_1, \ldots, \mathsf{ABE.ct}_\mathsf{N}\}$.
- For all $i \in [\mathsf{N}]$, set $s_i = \mathsf{ABE.Decrypt}(\mathsf{ABE.mpk}_i, \mathsf{sk}^{\mathsf{ABE}}_{\bar{\mathsf{x}},i}, \mathsf{ABE.ct}_i)$.
- Set $m := s_1 + \ldots + s_\mathsf{N}$.
- Output m.

– p-MA-ABE.$\mathsf{KeyRequest}(\mathsf{pk}, \mathsf{gid}, \bar{\mathsf{x}})$:
- For all $i \in [\mathsf{N}]$, parse mpk_i as $(\mathsf{Com.crs}_i, \mathsf{Sig.vk}_i, \mathsf{ABE.mpk}_i)$, compute $(c_i, d_i) \leftarrow \mathsf{Com.Commit}(\mathsf{Com.crs}_i, \mathsf{gid} \| \bar{\mathsf{x}})$ and set $\mathsf{req}_i := (\mathsf{gid}, c_i)$.
- Set $\mathsf{st} := (\mathsf{gid}, \bar{\mathsf{x}}, \{d_i\}_{i \in [\mathsf{N}]})$.
- Output $(\mathsf{st}, \{\mathsf{req}_i\}_{i \in [\mathsf{N}]})$.

– p-MA-ABE.$\mathsf{KeyGen}(t, \mathsf{msk}_t, \Theta_t^{\mathsf{gid}}, \mathsf{req}_t, \mathsf{pk})$: Computes
  - Parse msk as $(\mathsf{Sig.sk}_t, \mathsf{ABE.msk}_t)$.
  - For all $i \in [\mathsf{N}]$, parse mpk_i as $(\mathsf{Com.crs}_i, \mathsf{Sig.vk}_i, \mathsf{ABE.mpk}_i)$.
  - Parse req_t as (gid, c_t).
  - Fix $\pi_{\mathsf{pp}}^{(\alpha)} \in \mathcal{P}_{\text{p-MA-ABE}}$ where $\mathsf{pp} = (t, \mathsf{Com.crs}_t, c_t), \alpha = \Theta_t^{\mathsf{gid}}$, and $\mathsf{start} = \sigma^1_{\mathsf{pk},\mathsf{gid}}$.
  - Fix $\mu^{(\beta)} \in \mathcal{M}_{\text{p-MA-ABE}}$ where $\beta = (0, \mathsf{Sig.sk}_t, \mathsf{ABE.msk}_t)$.
  - Compute $\mathsf{sk}_{\mathsf{req}_t} \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi_{\mathsf{pp}}^{(\alpha)}, \mu^{(\beta)})$.
  - Output $\mathsf{sk}_{\mathsf{req}_t}$.

– p-MA-ABE.$\mathsf{KeyCombine}\left(\mathsf{st}, \{\mathsf{sk}_{\mathsf{req}_i}\}_{i\in[\mathsf{N}]}\right)$:
  - Parse st as $(\mathsf{gid}, \bar{\mathsf{x}}, \{d_i\}_{i\in[\mathsf{N}]})$.
  - For all $t \in [\mathsf{N}]$, parse $\mathsf{sk}_{\mathsf{req}_t}$ as ρ_t and evaluate as $\tau_t \leftarrow \rho_t(d_t)$[15]
  - Finally, for all $t \in [\mathsf{N}]$, compute $\mathsf{sk}_{\bar{\mathsf{x}},t}^{\mathsf{ABE}} \leftarrow \rho_t(\tau_1, \ldots, \tau_{\mathsf{N}})$.
  - Output $\{\mathsf{sk}_{\bar{\mathsf{x}},t}^{\mathsf{ABE}}\}_{t\in[\mathsf{N}]}$.

Fig. 17. A secure p-MA-ABE Protocol.

We now prove that the protocol in Figure 17 is in fact a secure p-MA-ABE scheme.

Lemma 6. *If there exists a CP-ABE scheme, a non-interactive UC secure commitment scheme, a puncturable signature scheme, and an R3PO scheme $\mathcal{O}_{p\text{-}MA\text{-}ABE}$ w.r.t. $(\mathcal{G}^1_{p\text{-}MA\text{-}ABE}, \mathcal{Q}_{p\text{-}MA\text{-}ABE})$[16], then there exists a secure p-MA-ABE scheme.*

Please refer to Appendix C for the full details of the proof.

References

[1] Shweta Agrawal, Rishab Goyal, and Junichi Tomida. "Multi-Party Functional Encryption". In: *Theory of Cryptography*. 2021.

[2] William Aiello, Yuval Ishai, and Omer Reingold. "Priced Oblivious Transfer: How to Sell Digital Goods". In: *EUROCRYPT*. 2001.

[3] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. *Differing-Inputs Obfuscation and Applications*. Cryptology ePrint Archive, Report 2013/689. 2013.

[4] Michael Backes, Birgit Pfitzmann, and Michael Waidner. "A General Composition Theorem for Secure Reactive Systems". In: *Theory of Cryptography*. 2004, pp. 336–354.

[5] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. "Obfuscation for Evasive Functions". In: *Theory of Cryptography*. 2014, pp. 26–51.

[6] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. "On the (Im)Possibility of Obfuscating Programs". In: *J. ACM* 59.2 (2012). ISSN: 0004-5411.

[7] Fabrice Benhamouda and Huijia Lin. "k-Round Multiparty Computation from k-Round Oblivious Transfer via Garbled Interactive Circuits". In: *EUROCRYPT*. Vol. 10821. Lecture Notes in Computer Science. 2018, pp. 500–532.

[8] Nir Bitansky, Ran Canetti, Yael Tauman Kalai, and Omer Paneth. "On Virtual Grey Box Obfuscation for General Circuits". In: *CRYPTO*. Vol. 8617. Lecture Notes in Computer Science. 2014, pp. 108–125.

[9] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. "Public Key Encryption with Keyword Search". In: *EUROCRYPT*. Ed. by Christian Cachin and Jan L. Camenisch. 2004, pp. 506–522.

[10] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. "Anonymous IBE, Leakage Resilience and Circular Security from New Assumptions". In: *EUROCRYPT*. 2018, pp. 535–564.

[11] Zvika Brakerski and Vinod Vaikuntanathan. "Lattice-Based FHE as Secure as PKE". In: ITCS '14. 2014, 1–12.

[12] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. *Encryption to the Future: A Paradigm for Sending Secret Messages to Future (Anonymous) Committees*. Cryptology ePrint Archive, Paper 2021/1423. 2021.

[15] formally, ρ_t also takes as input a state σ_1 and also outputs a state σ_2. for brevity, we ignore mentioning it here.

[16] which is also an R3PO w.r.t. $(\mathcal{G}^2_{\text{p-MA-ABE}}, \mathcal{Q}_{\text{p-MA-ABE}})$.

[13] R. Canetti. "Universally composable security: a new paradigm for cryptographic protocols". In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. 2001, pp. 136–145.
[14] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *J. Cryptol.* 13.1 (2000), 143–202.
[15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. "A Simpler Variant of Universally Composable Security for Standard Multiparty Computation". In: *CRYPTO*. 2015, pp. 3–22.
[16] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. "Obfuscation of Probabilistic Circuits and Applications". In: *Theory of Cryptography*. 2015, pp. 468–497.
[17] Melissa Chase. "Multi-authority Attribute Based Encryption". In: *Theory of Cryptography*. 2007, pp. 515–534.
[18] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. "Laconic Oblivious Transfer and Its Applications". In: *CRYPTO*. 2017, pp. 33–65.
[19] Pratish Datta, Ilan Komargodski, and Brent Waters. "Decentralized Multi-authority ABE for DNFs from LWE". In: *EUROCRYPT*. 2021, pp. 177–209.
[20] Danny Dolev, Cynthia Dwork, and Moni Naor. "Non-Malleable Cryptography". In: *SIAM Journal of Computing* 30 (Mar. 2001).
[21] Nico Döttling and Sanjam Garg. "From Selective IBE to Full IBE and Selective HIBE". In: *Theory of Cryptography*. 2017, pp. 372–408.
[22] Nico Döttling and Sanjam Garg. "Identity-Based Encryption from the Diffie-Hellman Assumption". In: *CRYPTO*. 2017, pp. 537–569.
[23] Cynthia Dwork, Moni Naor, and Amit Sahai. "Concurrent Zero-Knowledge". In: *J. ACM* 51.6 (2004), 851–898.
[24] Steven D. Galbraith and Lukas Zobernig. "Obfuscating Finite Automata". In: *SAC*. Vol. 12804. Lecture Notes in Computer Science. 2020, pp. 90–114.
[25] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. "Registration-Based Encryption: Removing Private-Key Generator from IBE". In: *Theory of Cryptography*. 2018, pp. 689–718.
[26] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. "Registration-Based Encryption from Standard Assumptions". In: *PKC*. 2019, pp. 63–93.
[27] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. *Black-Box Garbled RAM*. Cryptology ePrint Archive, Report 2015/307. 2015.
[28] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. "Garbled RAM From One-Way Functions". In: STOC '15. 2015, 449–458.
[29] Sanjam Garg and Antigoni Polychroniadou. "Two-Round Adaptively Secure MPC from Indistinguishability Obfuscation". In: *Theory of Cryptography*. 2015, pp. 614–637.
[30] Sanjam Garg and Akshayaram Srinivasan. "Two-Round Multiparty Secure Computation from Minimal Assumptions". In: *EUROCRYPT*. 2018, pp. 468–499.
[31] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. "Garbled RAM Revisited". In: *EUROCRYPT*. 2014, pp. 405–422.
[32] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. "Attribute-Based Encryption for Circuits". In: *J. ACM* 62.6 (2015).
[33] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. 2006, 89–98.
[34] Satoshi Hada and Toshiaki Tanaka. "On the existence of 3-round zero-knowledge protocols". In: *CRYPTO*. 1998, pp. 408–423.
[35] Dennis Hofheinz and Victor Shoup. "GNUC: A New Universal Composability Framework". In: *J. Cryptol.* 28.3 (2015), 423–508.
[36] Susan Hohenberger, Guy N. Rothblum, abhi shelat, and Vinod Vaikuntanathan. "Securely Obfuscating Re-encryption". In: *Theory of Cryptography*. 2007, pp. 233–252.
[37] Susan Hohenberger and Brent Waters. "Attribute-Based Encryption with Fast Decryption". In: *PKC*. 2013, pp. 162–179.
[38] Yuval Ishai, Omkant Pandey, and Amit Sahai. "Public-Coin Differing-Inputs Obfuscation and Its Applications". In: *Theory of Cryptography*. 2015, pp. 668–697.
[39] Aayush Jain, Huijia Lin, and Amit Sahai. "Indistinguishability Obfuscation from Well-Founded Assumptions". In: STOC 2021. 2021, 60–73.
[40] Charanjit S. Jutla and Arnab Roy. "Shorter Quasi-Adaptive NIZK Proofs for Linear Subspaces". In: *ASIACRYPT*. 2013, pp. 1–20.

[41] Yael Tauman Kalai. "Smooth Projective Hashing and Two-Message Oblivious Transfer". In: *EUROCRYPT*. 2005, pp. 78–95.
[42] Sam Kim. *Multi-Authority Attribute-Based Encryption from LWE in the OT Model*. Cryptology ePrint Archive, Report 2019/280. 2019.
[43] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. "Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption". In: *EUROCRYPT*. 2010, pp. 62–91.
[44] Allison Lewko and Brent Waters. "Decentralizing Attribute-Based Encryption". In: *EUROCRYPT*. 2011, pp. 568–588.
[45] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. "Indistinguishability Obfuscation with Non-trivial Efficiency". In: *PKC*. 2016, pp. 447–462.
[46] Steve Lu and Rafail Ostrovsky. "How to Garble RAM Programs?" In: *EUROCRYPT*. 2013, pp. 719–734.
[47] Benjamin Lynn, Manoj Prabhakaran, and Amit Sahai. "Positive Results and Techniques for Obfuscation". In: *EUROCRYPT*. 2004, pp. 20–39.
[48] Ueli Maurer. "Constructive Cryptography – A New Paradigm for Security Definitions and Proofs". In: *Theory of Security and Applications*. 2012, pp. 33–56.
[49] Daniele Micciancio and Stefano Tessaro. "An Equational Approach to Secure Multi-Party Computation". In: ITCS '13. 2013, 355–372.
[50] Yan Michalevsky and Marc Joye. "Decentralized Policy-Hiding ABE with Receiver Privacy". In: *ESORICS, Proceedings, Part II*. 2018, pp. 548–567.
[51] Moni Naor and Benny Pinkas. "Efficient oblivious transfer protocols". In: *SODA '01*. 2001.
[52] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. "A Framework for Efficient and Composable Oblivious Transfer". In: *CRYPTO*. 2008, pp. 554–571.
[53] Manoj Prabhakaran and Amit Sahai. "New Notions of Security: Achieving Universal Composability without Trusted Setup". In: STOC '04. 2004, 242–251.
[54] Amit Sahai and Brent Waters. "Fuzzy Identity-Based Encryption". In: *EUROCRYPT*. 2005, pp. 457–473.
[55] Amit Sahai and Brent Waters. "How to Use Indistinguishability Obfuscation: Deniable Encryption, and More". In: STOC '14. 2014, 475–484.
[56] Brent Waters, Hoeteck Wee, and David J. Wu. *Multi-Authority ABE from Lattices without Random Oracles*. Cryptology ePrint Archive, Paper 2022/1194. https://eprint.iacr.org/2022/1194. 2022. URL: https://eprint.iacr.org/2022/1194.
[57] Daniel Wichs and Giorgos Zirdelis. "Obfuscating Compute-and-Compare Programs under LWE". In: *FOCS*. 2017, pp. 600–611.
[58] Douglas Wikström. "Simplified Universal Composability Framework". In: *Theory of Cryptography*. 2016, pp. 566–595.
[59] Andrew Chi-Chih Yao. "How to generate and exchange secrets". In: *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. 1986, pp. 162–167.
[60] Kelly Yun, Xin Wang, and Rui Xue. "Identity-Based Functional Encryption for Quadratic Functions from Lattices". In: *Information and Communications Security*. 2018.

Appendix

A Proof of Theorem 1

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

Proof Sketch. The obfuscator $\mathcal{O}$ is shown in Figure 8. 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 $G \in \mathcal{G}$ and any $Q \in \mathcal{Q}$. For each $i \in \{1, 2, \cdots, N\}$, we define a sequence of hybrid experiments $\mathsf{Hybrid}^{i,0}$ through $\mathsf{Hybrid}^{i,6}$, and argue indistinguishability of the outputs from adjacent hybrids, and further show that the outcome of $\mathsf{Hybrid}^{i,6}$ is indistinguishable from that of $\mathsf{Hybrid}^{i+1,0}$. Finally, the outcome of the real obfuscation experiment will be shown to be identical to that of $\mathsf{Hybrid}^{1,0}$ and that of the ideal obfuscation experiment will be shown to be identical to that of $\mathsf{Hybrid}^{N+1,0}$.

For each $i \in [N]$, as we go from $\mathsf{Hybrid}^{i,0}$ to $\mathsf{Hybrid}^{i+1,0}$, 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 $\mathcal{E}_i$ 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 σ_i-restricted program, where $\{\sigma_i\}$ is as extracted by E_{i-1} and bounds the reach of the adversay in partition i.

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

– $\mathsf{Hybrid}^{i,1}$: In this hybrid, the i^{th} garbled circuit is simulated on an input 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 σ_i). Indistinguishability follows from a combination of reach restriction of E_{i-1} and garbled circuit security.

– $\mathsf{Hybrid}^{i,2}$: 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 $Q\hat{|}E_{i-1}\lfloor W_i$. J_i outputs a one-step σ_i-restricted reactive program, where σ_i is the state in partition i reachable by E_{i-1}. Indistinguishability holds since, the interaction between G and $Q\hat{|}E_{i-1}$ is unchanged, and so is the rest of the experiment.

– $\mathsf{Hybrid}^{i,3}$: In this hybrid, we use the decomposition of G to $\boldsymbol{\mathcal{L}}_i$ to have $H_i\|Z_i$ interact with $Q\hat{|}E_{i-1}\lfloor W_i$.

– $\mathsf{Hybrid}^{i,4}$: In this hybrid, we use the one-step extractor $\mathcal{E}_i$ and obfuscation simulator Sim_i for H_i from the library to simulate the $f_i(\sigma_i)$ (obfuscation of the one-step program at σ_i). This finishes the simulation of the i^{th} garbled circuit. Indistinguishability follows from R3PO security of $\mathcal{O}_i$.

– $\mathsf{Hybrid}^{i,5}$: In this hybrid, we rewrite the above as an interaction between $G\|J_i$ and $Q\hat{|}E_{i-1}\lfloor W_i\hat{|}\mathcal{E}_i$. Indistinguishability follows from the property of W_i, that it outputs its entire state as auxiliary output. Thus, decomposition holds even if a passive program $\mathcal{E}_i$ is allowed to see inside W_i.

– $\mathsf{Hybrid}^{i,6}$: 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 $E_{i-1}\lfloor W_i\hat{|}\mathcal{E}_i$ and J_i). Indistinguishability with $\mathsf{Hybrid}^{i,5}$ follows from the properties of J_i. Also hybrid $\mathsf{Hybrid}^{i,6}$ and $\mathsf{Hybrid}^{i+1,0}$ are identical to each other.

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

– Programs $\mathcal{J}_i$ and $\mathcal{K}_i$. Let the simulators corresponding to the decomposition (Definition 17) of G, Q to $\boldsymbol{\mathcal{L}}_i$ at any partition i be J_i, W_i, Z_i. We redefine J_i as a composition of two programs $\mathcal{J}_i$ and $\mathcal{K}_i$, s.t. $\mathcal{K}_i$ is a non-interacting program that uniformly samples $\widehat{\mu}_i^{(\widehat{\beta})}$ and outputs it at the end of the interaction. This corresponds to the fact that J_i samples $\widehat{\mu}_i^{(\widehat{\beta})}$ uniformly at the end of the interaction.

[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 $\mathcal{E}_k$

– Partial reach-extractor E_i. For any i, as we go from hybrid $\mathsf{Hybrid}^{i,0}$ to $\mathsf{Hybrid}^{i+1,0}$, 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 $\mathcal{E}_i$ as follows. E_{i-1} outputs $(\Pi^{i-1}, X^{*,i-1})$ and $\mathcal{E}_i$ outputs $(\widehat{\Pi_i}, \widehat{X_i^*})$; then, output of E_i is defined as $\Pi^i = \Pi^{i-1} \times \widehat{\Pi_i}$, $X^{*,i} = X^{*,i-1} \times \widehat{X_i^*}$. The partial extractor E_0 simply outputs a_Q.
– Sampling the labeling function $\widehat{\beta}$.
  - Hybrids with generator G sample the labeling function $\widehat{\beta}$ uniformly at random outside the interaction.
  - Hybrids with generator $G[\![\mathcal{J}_i\|\mathcal{K}_i$ has $\mathcal{K}_i$ sample $\widehat{\beta}(k,.,.)$ for $k \in [i,N]$ uniformly at random. The labeling functions $\widehat{\beta}(k,.,.)$ for $k < i$ are sampled uniformly at random outside the interaction.
  - Hybrids with generator $H_i[\![Z_i$ has Z_i sample $\widehat{\beta}(k,.,.)$ for $k \in [i,N]$ uniformly at random. The labeling functions $\widehat{\beta}(k,.,.)$ for $k < i$ are sampled uniformly at random outside the interaction.

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

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 $\mathcal{G}$ w.r.t. the adversary Q corresponds to E_{N+1}. Similarly, the R3PO simulator is essentially $\mathsf{Hybrid}^{N+1,0}$, excluding the interaction.

- $\mathsf{Hybrid}^{i,0}$: In this hybrid, the reactive program is generated by G while interacting with the adversary $Q\hat{|}E_{i-1}$, where E_{i-1} is empty for $i=1$ and the composite machine $(E_{i-2}|W_{i-1}\hat{|}\mathcal{E}_i\|J_{i-1})$ for $i>1$. We show in Lemma 7 that E_{i-1} is a i-partial reach-extractor for Q w.r.t. G.

  $\mathsf{Hybrid}^{i,0}$:
  – $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G;\ a_Q, \Pi^{i-1}, X^{*,i-1}) \leftarrow \langle G : Q\hat{|}E_{i-1}\rangle$
  – Sample $\widehat{\beta}$.
  – Define $\Sigma_{rch} = \text{REACH}_{\Pi^{i-1}}(X^{*,i-1})$ and extract $\{\widehat{\Pi_j}, \widehat{X_j^*})\}_{1\le j<i}$ from $(\Pi^{i-1}, X^{*,i-1})$.
  – For each $j \in [N]$, define functions $\widehat{\mu}_j^{(\widehat{\beta})}, f_j$ as follows:
    - $\forall \sigma' \in \Sigma,\quad \widehat{\mu}_j^{(\widehat{\beta})}(\sigma') := \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j}(\sigma')$
    - $\forall \sigma \in \Sigma,\quad f_j(\sigma) := \Big(\mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})}),\ \mu^{(\beta)}(\sigma)\Big)$
  – For each $j \in [N]$ such that $j < i$:
    - $\sigma_j = \Sigma_{rch} \cap \Sigma_j$
    - If $\sigma_j = \perp$, then $\widetilde{\text{GC}_j} \leftarrow \mathsf{GCircSim}\Big(\Phi(f_j)\Big)$
    - Else:
      - Sample $\widetilde{\rho_j} \leftarrow \mathsf{Sim}_j\big(\widehat{\pi}_{\sigma_j}^{(\cdot)}, \widehat{\mu}_j^{(\cdot)}, \widehat{\Pi_j}, \{x, \widehat{\mu}_j^{(\widehat{\beta})}(\text{REACH}_{\widehat{\Pi_j}}(x))\}_{x\in\widehat{X_j^*}}\big)$
      - Let $y_{\sigma_j} = (\widetilde{\rho_j},\ \mu^{(\beta)}(\sigma_j))$, $lab_{\sigma_j} = \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j-1}(\sigma_j)$
      - If $j>1$, sample $\widetilde{\text{GC}_j} \leftarrow \mathsf{GCSim}\Big(\Phi(f_j), lab_{\sigma_j}, y_{\sigma_j}\Big)$
  – For each $1 < j \le N$ such that $j \ge i$, sample $\text{GC}_j \leftarrow \mathsf{GCGarble}(f_j, \widehat{\beta}_j)$
  – If $i \in \{1,2\}$, set $\Big(y_{\sigma_1}, \{\text{GC}_j\}_{1<j\le N}\Big)$
    else $\rho = \Big(y_{\sigma_1}, \{\widetilde{\text{GC}_j}\}_{1<j<i}, \{\text{GC}_j\}_{N\ge j\ge i}\Big)$
  – Output (a_G, a_Q, ρ)

- $\mathsf{Hybrid}^{i,1}$: In this hybrid, the i^{th} garbled circuit is simulated given a input σ_i, input labels $\mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},i}(\sigma_i)$ 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* $\mathsf{Hybrid}^{i,0}$*:* Note that, from the reach bound and reach restriction of E_{i-1}, an adversary can evaluate the garbled circuit GC_i in $\mathsf{Hybrid}^{i,0}$ at most a single point (corresponding to the state reached: σ_i). Thus, from a reduction to garbled circuit security, the indistinguishability follows.

$\mathsf{Hybrid}^{i,1}$:

- $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G;\ a_Q, \Pi^{i-1}, X^{*,i-1}) \leftarrow \langle G : Q \hat{|} E_{i-1} \rangle$
- Sample $\widehat{\beta}$.
- Define $\Sigma_{rch} = \text{REACH}_{\Pi^{i-1}}(X^{*,i-1})$ and extract $\{\widehat{\Pi_j}, \widehat{X_j^*})\}_{1 \le j < i}$ from $(\Pi^{i-1}, X^{*,i-1})$.
- For each $j \in [N]$, define functions $\widehat{\mu}_j^{(\widehat{\beta})}, f_j$ as follows:
  - $\forall \sigma' \in \Sigma, \quad \widehat{\mu}_j^{(\widehat{\beta})}(\sigma') := \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j}(\sigma')$
  - $\forall \sigma \in \Sigma, \quad f_j(\sigma) := \left(\mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})}),\ \mu^{(\beta)}(\sigma)\right)$
- For each $j \in [N]$ such that $j \le i$:
  - $\sigma_j = \Sigma_{rch} \cap \Sigma_j$
  - If $\sigma_j = \perp$, then $\widetilde{\text{GC}_j} \leftarrow \mathsf{GCircSim}\left(\Phi(f_j)\right)$
  - Else:
    - If $j < i$,
      sample $\widetilde{\rho_j} \leftarrow \mathsf{Sim}_j\left(\widehat{\pi}_{\sigma_j}^{(\cdot)}, \widehat{\mu}_j^{(\cdot)}, \widehat{\Pi_j}, \{x, \widehat{\mu}_j^{(\widehat{\beta})}(\text{REACH}_{\widehat{\Pi_j}}(x))\}_{x \in \widehat{X_j^*}}\right)$
      and set $y_{\sigma_j} = (\widetilde{\rho_j},\ \mu^{(\beta)}(\sigma_j))$,
    - If $j = i$, set $\rho_j = \mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})})$ and $y_{\sigma_j} = (\rho_j,\ \mu^{(\beta)}(\sigma_j))$
    - If $j \le i$, $lab_{\sigma_j} = \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j-1}(\sigma_j)$
    - If $j > 1$, sample $\widetilde{\text{GC}_j} \leftarrow \mathsf{GCSim}\left(\Phi(f_j), lab_{\sigma_j}, y_{\sigma_j}\right)$
- For each $j \in [N]$ such that $j > i$, sample $\text{GC}_j \leftarrow \mathsf{GCGarble}(f_j, \widehat{\beta}_j)$
- If $i \in \{1, 2\}$, set $\left(y_{\sigma_1}, \{\text{GC}_j\}_{1 < j \le N}\right)$
  else $\rho = \left(y_{\sigma_1}, \{\widetilde{\text{GC}_j}\}_{1 < j < i}, \{\text{GC}_j\}_{N \ge j \ge i}\right)$
- Output (a_G, a_Q, ρ)

- $\mathsf{Hybrid}^{i,2}$: In this hybrid, we let $G \| \mathcal{J}_i | \mathcal{K}_i$ interact with $Q \hat{|} E_{i-1} | W_i$, where $\mathcal{J}_i$, $\mathcal{K}_i$, W_i are as specified by the decomposition of G to $\boldsymbol{\mathcal{L}}_i$ at partition i. *Indistinguishability with* $\mathsf{Hybrid}^{i,1}$*:* $\mathcal{J}_i$, $\mathcal{K}_i$ and W_i are such that, they are passive to the interaction between G and Q. $\mathcal{K}_i$ uniformly samples $\widehat{\beta}(k, ., .)$ for $k \in [i, N]$, but only outputs it at the end of the interaction. Thus, we have that the distributions of $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G, \widehat{\beta})$ and output of E_{i-1} are identical in both the hybrids. Hence the hybrids are identical.
  From the correctness of decomposition (Definition 17), $\mathcal{J}_i$ outputs a one-step restriction at state σ_i consistent with the extraction of E_{i-1} (which is a i-partial reach-extractor).

$\mathsf{Hybrid}^{i,2}$:

- $(\widehat{\pi}_{\sigma_i}^{(\alpha)}, \widehat{\mu}_{\sigma_i}^{(\widehat{\beta})}, \pi^{(\alpha)}, \mu^{(\beta)}, a_G;\ a_Q, \Pi^{i-1}, X^{*,i-1}) \leftarrow \langle G \| \mathcal{J}_i \| \mathcal{K}_i : Q \hat{|} E_{i-1} | W_i \rangle$
- Sample $\widehat{\beta}$ consistent with $\widehat{\mu}_{\sigma_i}^{(\widehat{\beta})}$ (that is, sample $\widehat{\beta}(k,.,.)$ for $k < i$).
- Define $\Sigma_{rch} = \textsc{reach}_{\Pi^{i-1}}(X^{*,i-1})$ and extract $\{\widehat{\Pi_j}, \widehat{X_j^*})\}_{1 \le j < i}$ from $(\Pi^{i-1}, X^{*,i-1})$.
- For each $j \in [N]$, define functions $\widehat{\mu}_j^{(\widehat{\beta})}, f_j$ as follows:
  - $\forall \sigma' \in \Sigma, \quad \widehat{\mu}_j^{(\widehat{\beta})}(\sigma') := \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j}(\sigma')$
  - $\forall \sigma \in \Sigma, \quad f_j(\sigma) := \Big(\mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})}),\ \mu^{(\beta)}(\sigma)\Big)$
- For each $j \in [N]$ such that $j \le i$:
  - $\sigma_j = \Sigma_{rch} \cap \Sigma_j$
  - If $\sigma_j = \perp$, then $\widetilde{\mathrm{GC}_j} \leftarrow \mathsf{GCircSim}\Big(\Phi(f_j)\Big)$
  - Else:
    - If $j < i$,
      sample $\widetilde{\rho_j} \leftarrow \mathsf{Sim}_j\big(\widehat{\pi}_{\sigma_j}^{(\cdot)}, \widehat{\mu}_j^{(\cdot)}, \widehat{\Pi_j}, \{x, \widehat{\mu}_j^{(\widehat{\beta})}(\textsc{reach}_{\widehat{\Pi_j}}(x))\}_{x \in \widehat{X_j^*}}\big)$
      and set $y_{\sigma_j} = (\widetilde{\rho_j},\ \mu^{(\beta)}(\sigma_j))$,
    - If $j = i$, set $\rho_j = \mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})})$ and $y_{\sigma_j} = (\rho_j,\ \mu^{(\beta)}(\sigma_j))$
    - If $j \le i$, $lab_{\sigma_j} = \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j-1}(\sigma_j)$
    - If $j > 1$, sample $\widetilde{\mathrm{GC}_j} \leftarrow \mathsf{GCSim}\Big(\Phi(f_j), lab_{\sigma_j}, y_{\sigma_j}\Big)$
- For each $j \in [N]$ such that $j > i$, sample $\mathrm{GC}_j \leftarrow \mathsf{GCGarble}(f_j, \widehat{\beta}_j)$
- If $i \in \{1, 2\}$, set $\Big(y_{\sigma_1}, \{\mathrm{GC}_j\}_{1 < j \le N}\Big)$
  else $\rho = \Big(y_{\sigma_1}, \{\widetilde{\mathrm{GC}_j}\}_{1 < j < i}, \{\mathrm{GC}_j\}_{N \ge j \ge i}\Big)$
- Output (a_G, a_Q, ρ)

- $\mathsf{Hybrid}^{i,3}$: In this hybrid, we use the interaction $\langle H_i \| Z_i : Q \hat{|} E_{i-1} | W_i \rangle$ to generate the reactive program, where H_i corresponds to the decomposition of G to $\boldsymbol{\mathcal{L}}_i$. *Indistinguishability with* $\mathsf{Hybrid}^{i,2}$ trivially follows: corresponding to Q, E_{i-1}, $\mathcal{J}_i$, $\mathcal{K}_i$, W_i of the previous hybrid, Z_i is such that the distributions $\langle G \| \mathcal{J}_i \| \mathcal{K}_i : Q \hat{|} E_{i-1} | W_i \rangle$ and $\langle H_i \| Z_i : Q \hat{|} E_{i-1} | W_i \rangle$ are indistinguishable.

$\mathsf{Hybrid}^{i,3}$:

- $(\widehat{\pi}_{\sigma_i}^{(\alpha)}, \widehat{\mu}_{\sigma_i}^{(\widehat{\beta})}, a_{Z_i};\ a_Q, \Pi^{i-1}, X^{*,i-1}) \leftarrow \langle H_i \| Z_i : Q \hat{|} E_{i-1} | W_i \rangle$
- Parse a_{Z_i} as $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G)$.
- Sample $\widehat{\beta}$ consistent with $\widehat{\mu}_{\sigma_i}^{(\widehat{\beta})}$ (that is, sample $\widehat{\beta}(k,.,.)$ for $k < i$).
- Define $\Sigma_{rch} = \text{REACH}_{\Pi^{i-1}}(X^{*,i-1})$ and extract $\{\widetilde{\Pi_j}, \widehat{X_j^*})\}_{1 \le j < i}$ from $(\Pi^{i-1}, X^{*,i-1})$.
- For each $j \in [N]$, define functions $\widehat{\mu}_j^{(\widehat{\beta})}, f_j$ as follows:
  - $\forall \sigma' \in \Sigma, \quad \widehat{\mu}_j^{(\widehat{\beta})}(\sigma') := \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j}(\sigma')$
  - $\forall \sigma \in \Sigma, \quad f_j(\sigma) := \Big(\mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})}),\ \mu^{(\beta)}(\sigma)\Big)$
- For each $j \in [N]$ such that $j \le i$:
  - $\sigma_j = \Sigma_{rch} \cap \Sigma_j$
  - If $\sigma_j = \perp$, then $\widetilde{\text{GC}_j} \leftarrow \mathsf{GCircSim}\Big(\Phi(f_j)\Big)$
  - Else:
    - If $j < i$,
      sample $\widetilde{\rho_j} \leftarrow \mathsf{Sim}_j\big(\widehat{\pi}_{\sigma_j}^{(\cdot)}, \widehat{\mu}_j^{(\cdot)}, \widehat{\Pi_j}, \{x, \widehat{\mu}_j^{(\widehat{\beta})}(\text{REACH}_{\widehat{\Pi_j}}(x))\}_{x \in \widehat{X_j^*}}\big)$
      and set $y_{\sigma_j} = (\widetilde{\rho_j},\ \mu^{(\beta)}(\sigma_j))$,
    - If $j = i$, set $\rho_j = \mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})})$ and $y_{\sigma_j} = (\rho_j,\ \mu^{(\beta)}(\sigma_j))$
    - If $j \le i$, $lab_{\sigma_j} = \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j-1}(\sigma_j)$
    - If $j > 1$, sample $\widetilde{\text{GC}_j} \leftarrow \mathsf{GCSim}\Big(\Phi(f_j), lab_{\sigma_j}, y_{\sigma_j}\Big)$
- For each $j \in [N]$ such that $j > i$, sample $\text{GC}_j \leftarrow \mathsf{GCGarble}(f_j, \widehat{\beta}_j)$
- If $i \in \{1, 2\}$, set $\Big(y_{\sigma_1}, \{\text{GC}_j\}_{1 < j \le N}\Big)$
  else $\rho = \Big(y_{\sigma_1}, \{\widetilde{\text{GC}_j}\}_{1 < j < i}, \{\text{GC}_j\}_{N \ge j \ge i}\Big)$
- Output (a_G, a_Q, ρ)

- $\mathsf{Hybrid}^{i,4}$: In this hybrid, we let $H_i \| Z_i$ interact with $Q \hat{|} E_{i-1} | W_i \hat{|} \mathcal{E}_i$, where $\mathcal{E}_i$ is a reach-extractor for $Q \hat{|} E_{i-1} | W_{i-1}$ (such an extractor is guaranteed from the R3PO for $\boldsymbol{\mathcal{L}}_i$). Further, the i^{th} one-step obfuscation program is simulated using the R3PO simulator for $\mathcal{H}_i$ and $\mathcal{O}_i$. *Indistinguishability with* $\mathsf{Hybrid}^{i,3}$*:* directly follows from from the one-step extractor guarantee and R3PO security of the one-step generator $H_i \| Z_i$ (Definition 16).

$\mathsf{Hybrid}^{i,4}$:

- $(\widehat{\pi}_{\sigma_i}^{(\alpha)}, \widehat{\mu}_{\sigma_i}^{(\widehat{\beta})}, a_{Z_i};\ a_Q, \Pi^{i-1}, X^{*,i-1}, (\widehat{\Pi_i}, \widehat{X_i^*})) \leftarrow \langle H_i \| Z_i : Q \hat{|} E_{i-1} | W_i \hat{|} \mathcal{E}_i \rangle$
- Parse a_{Z_i} as $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G)$.
- Sample $\widehat{\beta}$ consistent with $\widehat{\mu}_{\sigma_i}^{(\widehat{\beta})}$ (that is, sample $\widehat{\beta}(k,.,.)$ for $k < i$).
- Define $\Sigma_{rch} = \textsc{reach}_{\Pi^{i-1}}(X^{*,i-1}) \cup \textsc{reach}_{\widehat{\Pi_i}}(\widehat{X_i^*})$ and extract $\{\widehat{\Pi_j}, \widehat{X_j^*})\}_{1 \le j < i}$ from $(\Pi^{i-1}, X^{*,i-1})$.
- For each $j \in [N]$, define functions $\widehat{\mu}_j^{(\widehat{\beta})}, f_j$ as follows:
  - $\forall \sigma' \in \Sigma, \quad \widehat{\mu}_j^{(\widehat{\beta})}(\sigma') := \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j}(\sigma')$
  - $\forall \sigma \in \Sigma, \quad f_j(\sigma) := \Big(\mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})}),\ \mu^{(\beta)}(\sigma)\Big)$
- For each $j \in [N]$ such that $j \le i$:
  - $\sigma_j = \Sigma_{rch} \cap \Sigma_j$
  - If $\sigma_j = \perp$, then $\widetilde{\mathrm{GC}_j} \leftarrow \mathsf{GCircSim}\Big(\Phi(f_j)\Big)$
  - Else:
    - Sample $\widetilde{\rho_j} \leftarrow \mathsf{Sim}_j\big(\widehat{\pi}_{\sigma_j}^{(\cdot)}, \widehat{\mu}_j^{(\cdot)}, \widehat{\Pi_j}, \{x, \widehat{\mu}_j^{(\widehat{\beta})}(\textsc{reach}_{\widehat{\Pi_j}}(x))\}_{x \in \widehat{X_j^*}}\big)$ and set $y_{\sigma_j} = (\widetilde{\rho_j},\ \mu^{(\beta)}(\sigma_j)),\ lab_{\sigma_j} = \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j-1}(\sigma_j)$
    - If $j > 1$, sample $\widetilde{\mathrm{GC}_j} \leftarrow \mathsf{GCSim}\Big(\Phi(f_j), lab_{\sigma_j}, y_{\sigma_j}\Big)$
- For each $j \in [N]$ such that $j > i$, sample $\mathrm{GC}_j \leftarrow \mathsf{GCGarble}(f_j, \widehat{\beta}_j)$
- If $i \in \{1,2\}$, set $\Big(y_{\sigma_1}, \{\mathrm{GC}_j\}_{1 < j \le N}\Big)$ else $\rho = \Big(y_{\sigma_1}, \{\widetilde{\mathrm{GC}_j}\}_{1<j<i}, \{\mathrm{GC}_j\}_{N \ge j \ge i}\Big)$
- Output (a_G, a_Q, ρ)

- $\mathsf{Hybrid}^{i,5}$: In this hybrid, we re-invoke the $\mathcal{J}_i$, $\mathcal{K}_i$ (used in the previous hybrids). *Indistinguishability with* $\mathsf{Hybrid}^{i,4}$*:* We prove this via a reduction to the decomposition guarantee. Suppose the hybrids were distinguishable, then there exists a distinguisher (that simulates $\mathcal{E}_i$ using the auxiliary output of W_i, which contains its entire state and view) and can break the decomposition.

$\mathsf{Hybrid}^{i,5}$:

- $(\widehat{\pi}_{\sigma_i}^{(\alpha)}, \widehat{\mu}_{\sigma_i}^{(\widehat{\beta})}, \pi^{(\alpha)}, \mu^{(\beta)}, a_G, ;\ a_Q, \Pi^{i-1}, X^{*,i-1}, (\widehat{\Pi_i}, \widehat{X_i^*})) \leftarrow \langle G \| \mathcal{J}_i \| \mathcal{K}_i : Q \hat{|} E_{i-1} | W_i \hat{|} \mathcal{E}_i \rangle$
- Sample $\widehat{\beta}$ consistent with $\widehat{\mu}_{\sigma_i}^{(\widehat{\beta})}$ (that is, sample $\widehat{\beta}(k,.,.)$ for $k < i$).
- Define $\Sigma_{rch} = \textsc{reach}_{\Pi^{i-1}}(X^{*,i-1}) \cup \textsc{reach}_{\widehat{\Pi_i}}(\widehat{X_i^*})$ and extract $\{\widehat{\Pi_j}, \widehat{X_j^*})\}_{1 \le j < i}$ from $(\Pi^{i-1}, X^{*,i-1})$.
- For each $j \in [N]$, define functions $\widehat{\mu}_j^{(\widehat{\beta})}, f_j$ as follows:
  - $\forall \sigma' \in \Sigma, \quad \widehat{\mu}_j^{(\widehat{\beta})}(\sigma') := \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j}(\sigma')$
  - $\forall \sigma \in \Sigma, \quad f_j(\sigma) := \Big(\mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})}),\ \mu^{(\beta)}(\sigma)\Big)$
- For each $j \in [N]$ such that $j \le i$:
  - $\sigma_j = \Sigma_{rch} \cap \Sigma_j$
  - If $\sigma_j = \perp$, then $\widetilde{\mathrm{GC}_j} \leftarrow \mathsf{GCircSim}\Big(\Phi(f_j)\Big)$
  - Else:
    - Sample $\widetilde{\rho_j} \leftarrow \mathsf{Sim}_j\big(\widehat{\pi}_{\sigma_j}^{(\cdot)}, \widehat{\mu}_j^{(\cdot)}, \widehat{\Pi_j}, \{x, \widehat{\mu}_j^{(\widehat{\beta})}(\textsc{reach}_{\widehat{\Pi_j}}(x))\}_{x \in \widehat{X_j^*}}\big)$ and set $y_{\sigma_j} = (\widetilde{\rho_j},\ \mu^{(\beta)}(\sigma_j))$, $lab_{\sigma_j} = \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j-1}(\sigma_j)$
    - If $j > 1$, sample $\widetilde{\mathrm{GC}_j} \leftarrow \mathsf{GCSim}\Big(\Phi(f_j), lab_{\sigma_j}, y_{\sigma_j}\Big)$
- For each $j \in [N]$ such that $j > i$, sample $\mathrm{GC}_j \leftarrow \mathsf{GCGarble}(f_j, \widehat{\beta}_j)$
- If $i \in \{1,2\}$, set $\Big(y_{\sigma_1}, \{\mathrm{GC}_j\}_{1<j\le N}\Big)$ else $\rho = \Big(y_{\sigma_1}, \{\widetilde{\mathrm{GC}_j}\}_{1<j<i}, \{\mathrm{GC}_j\}_{N \ge j \ge i}\Big)$
- Output (a_G, a_Q, ρ)

- $\mathsf{Hybrid}^{i,6}$: We now move the sampling to outside the interaction (that is, remove $\mathcal{K}_i$). It remains indistinguishable since $\mathcal{K}_i$ was non-interacting and producing its output at the end of the interaction. Finally, we rewrite the interaction as $\langle G : Q \hat{|} E_{i-1} | W_i \hat{|} \mathcal{E}_i \| \mathcal{J}_i \rangle$. *Indistinguishability with* $\mathsf{Hybrid}^{i,5}$*:* this follows from the description of $\mathcal{J}_i$, which we defined as only being allowed to see the interaction of $\langle G : Q \rangle$ (and not the output of G). *Indistinguishability with* $\mathsf{Hybrid}^{i+1,0}$*:* we define $E_i = E_{i-1} | W_i \hat{|} \mathcal{E}_i \| J_i$. Then, the hybrids are identical.

$\mathsf{Hybrid}^{i,6}$:

- $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G;\ a_Q, \Pi^{i-1}, X^{*,i-1}, (\widehat{\Pi}_i, \widehat{X}_i^*)) \leftarrow \langle G : Q\hat{|}E_{i-1}|W_i\hat{|}\mathcal{E}_i\|J_i\rangle$
- Sample $\widehat{\beta}$.
- Define $\Sigma_{rch} = \textsc{reach}_{\Pi^{i-1}}(X^{*,i-1}) \cup \textsc{reach}_{\widehat{\Pi}_i}(\widehat{X}_i^*)$ and extract $\{\widehat{\Pi}_j, \widehat{X}_j^*)\}_{1\le j<i}$ from $(\Pi^{i-1}, X^{*,i-1})$.
- For each $j \in [N]$, define functions $\widehat{\mu}_j^{(\widehat{\beta})}, f_j$ as follows:
  - $\forall \sigma' \in \Sigma,\quad \widehat{\mu}_j^{(\widehat{\beta})}(\sigma') := \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j}(\sigma')$
  - $\forall \sigma \in \Sigma,\quad f_j(\sigma) := \Big(\mathcal{O}_j(\widehat{\pi}_\sigma^{(\alpha)}, \widehat{\mu}_j^{(\widehat{\beta})}),\ \mu^{(\beta)}(\sigma)\Big)$
- For each $j \in [N]$ such that $j \le i$:
  - $\sigma_j = \Sigma_{rch} \cap \Sigma_j$
  - If $\sigma_j = \perp$, then $\widetilde{\mathrm{GC}_j} \leftarrow \mathsf{GCircSim}\Big(\Phi(f_j)\Big)$
  - Else:
    - Sample $\widetilde{\rho_j} \leftarrow \mathsf{Sim}_j\big(\widehat{\pi}_{\sigma_j}^{(\cdot)}, \widehat{\mu}_j^{(\cdot)}, \widehat{\Pi}_j, \{x, \widehat{\mu}_j^{(\widehat{\beta})}(\textsc{reach}_{\widehat{\Pi}_j}(x))\}_{x\in\widehat{X}_j^*}\big)$ and set $y_{\sigma_j} = (\widetilde{\rho_j},\ \mu^{(\beta)}(\sigma_j))$, $lab_{\sigma_j} = \mathsf{encode}_{\widehat{\beta}}^{\mathcal{I},j-1}(\sigma_j)$
    - If $j > 1$, sample $\widetilde{\mathrm{GC}_j} \leftarrow \mathsf{GCSim}\Big(\Phi(f_j), lab_{\sigma_j}, y_{\sigma_j}\Big)$
- For each $j \in [N]$ such that $j > i$, sample $\mathrm{GC}_j \leftarrow \mathsf{GCGarble}(f_j, \widehat{\beta}_j)$
- If $i \in \{1, 2\}$, set $\Big(y_{\sigma_1}, \{\mathrm{GC}_j\}_{1<j\le N}\Big)$ else $\rho = \Big(y_{\sigma_1}, \{\widetilde{\mathrm{GC}_j}\}_{1<j<i}, \{\mathrm{GC}_j\}_{N\ge j\ge i}\Big)$
- Output (a_G, a_Q, ρ)

Lemma 7. *For each $i \in [N+1]$, E_{i-1} is an i-partial reach extractor for G and Q with all but negligible probability.*

Proof: The claim trivially holds for $i = 1$. Now, we show that for any $i \in [N]$, if E_{i-1} is an i-partial reach extractor then E_i is an $(i+1)$-partial reach extractor, which completes the proof. Recall that, E_{i-1} outputs $(\Pi^{i-1},\ X^{*,i-1})$; the one-step extractor $\mathcal{E}_i$ outputs $(\widehat{\Pi}_i,\ \widehat{X}_i^*)$; finally, E_i is of the form $E_{i-1}|W_i|\mathcal{E}_i|\mathcal{J}_i$ and outputs $\Pi^i = \Pi^{i-1} \times \widehat{\Pi}_i$, $X^{*,i} = X^{*,i-1} \times \widehat{X}_i^*$.
Now, from the tree-ordering property of the reactive programs output by G, there is at most a single partition $k \le i$ from which states in partition $i+1$ can be reached. If $k = i$, then reach-restriction and reach-bounding holds from the security of the one-step extractor $\mathcal{E}_i$, if $k < i$, then reach-restriction and reach-bounding holds from the security of E_{i-1}. Thus, E_i as defined above, is a $(i+1)$-partial reach-extractor. □

□

A.1 Randomized message functions.

The message function $\mu^{(\beta)}$ could, in general, be a randomized function. But, due to the reach-restriction requirement, one can replace it (at the time of obfuscation) with a deterministic function which uses N hard-coded random tapes, one each for evaluating $\mu^{(\beta)}$ on all the states in a partition. In both cases, a simulator Sim gets the same message outputs (corresponding to the reachable states). Then, if R3PO security holds for the first message function, it must also hold for the second (with the same simulator). Thus, w.l.o.g, for the rest of the paper, we will only consider deterministic message functions.

B Details Omitted from Section 6

B.1 Commitment-Opening R3PO

Let generator class and adversary class be as described in Section 6.1. We show that a commitment scheme Com and an R3PO scheme w.r.t. $(\mathcal{G}_{\textsc{commit}}^{\mathsf{Com},\mathcal{M}},\ \mathcal{Q}_{\textsc{commit}}^{\mathsf{Com}},\ \mathring{\mathcal{P}}_{\textsc{commit}})$ 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 $(\mathsf{Commit}, \mathsf{Open})$ 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. [18] 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).

Commitment Scheme from OT

Security Parameter: Let κ be the security parameter.
Let $\mathsf{OT} = (\mathsf{OT.Setup}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ be an OT scheme.

Commitment Scheme Com**:**

- $\mathsf{Com.Setup}(1^\kappa)$: Run l instances of the $\mathsf{OT.Setup}(1^\kappa)$ algorithm, i.e.,

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

  and output $\mathsf{Com.crs} := \{\mathsf{OT.crs}^i\}_{i\in[l]}$.
- $\mathsf{Com.Commit}(\mathsf{Com.crs}, x)$: Parse $\mathsf{Com.crs} = \{\mathsf{OT.crs}^i\}_{i\in[l]}$. Commit each bit of x as:

$$\forall i \in [l],\ (\mathsf{ots}_1^i, \omega^i) \leftarrow \mathsf{OT}_1(\mathsf{OT.crs}^i, x_i)$$

  and output $(c, d) := \left(\{\mathsf{ots}_1^i\}_{i\in[l]},\ \{\omega^i\}_{i\in[l]}\right)$.
- $\mathsf{Com.Open}(\mathsf{Com.crs}, c, d)$: Parse $(\mathsf{Com.crs}, c, d)$ as $\left(\{\mathsf{OT.crs}^i\}_{i\in[l]},\ \{\mathsf{ots}_1^i\}_{i\in[l]},\ \{\omega^i\}_{i\in[l]}\right)$. Open each bit of x as follows. For all $i \in [l]$:

$$(m_0, m_1) \leftarrow \{0,1\}^\kappa,$$
$$\mathsf{ots}_2 \leftarrow \mathsf{OT}_2(\mathsf{OT.crs}^i, \mathsf{ots}_1^i, m_0, m_1),$$
$$m \leftarrow \mathsf{OT}_3(\mathsf{OT.crs}^i,\ \mathsf{ots}_2,\ \omega^i),$$
$$x_i = 0 \text{ if } m = m_0, \text{ else } x_i = 1$$

  and output x.

Fig. 18. Commitment from OT

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

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

Proof: We prove this for the message function class $\widehat{\mathcal{M}}$ (Definition 18). 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 $\widehat{\mathcal{M}}$. We give the proof for the UC-secure version. In particular, we use a UC-secure 2-round OT scheme (Definition 5) to give a commitment scheme and a R3PO scheme w.r.t. $(\mathcal{G}^{\mathsf{Com}}_{\mathrm{COMMIT}}, \mathcal{Q}^{\mathsf{Com}}_{\mathrm{COMMIT}}, \mathring{\mathcal{P}}_{\mathrm{COMMIT}})$. It is easy to show that the scheme in Figure 18 is a valid commitment scheme (Definition 3). We now prove that obfuscator $\mathcal{O}^{\mathsf{Com}}_{\mathrm{COMMIT}}$ in Figure 19 is a secure R3PO.

[18] Formally, the weak binding property is that for all PPT $\mathcal{A}$, and $\forall m \in \mathsf{M}$, $\Pr[(c,d) \leftarrow \mathsf{Commit}(m;r), \mathsf{Open}(c, \mathcal{A}(m,r)) = m', m' \notin \{m, \perp\}]$ 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 $\forall m_1, m_2 \in \mathsf{M}$, $\mathsf{Commit}(m_1)$ and $\mathsf{Commit}(m_2)$ are computationally indistinguishable.

– **Correctness.** If input d satisfies $\mathsf{Com.Open}(\mathsf{crs}, c, d) = x$, then from the correctness of the OT scheme, the evaluator can recover function values $\{\widehat{\beta}(2, i, x_i)\}_{i\in[l]}$ for x, which is equal to $\widehat{\mu}^{(\widehat{\beta})}(\sigma_x)$.
– **Efficiency.** Follows from the efficiency of (l instances of) OT scheme.
– **Security.** We need to provide a $\mathsf{Sim}^{\mathsf{Com}}_{\text{COMMIT}}$ s.t. $\forall\ G \in \mathcal{G}^{\mathsf{Com}}_{\text{COMMIT}}$ and $\forall Q \in \mathcal{Q}^{\mathsf{Com}}_{\text{COMMIT}}$, there exists a reach-extractor $\mathcal{E}^{\mathsf{Com}}_{\text{COMMIT}}$ w.r.t. $(\mathcal{G}^{\mathsf{Com}}_{\text{COMMIT}}, \mathring{\mathcal{P}}_{\text{COMMIT}})$ which satisfies Definition 16.
  - $\mathcal{E}^{\mathsf{Com}}_{\text{COMMIT}}$:
    * Let $(\mathcal{E}_1, \mathcal{E}_2)$ be the extractors from the binding property of Com. $\mathcal{E}^{\mathsf{Com}}_{\text{COMMIT}}$ configures the crs as $(\mathsf{crs}, \mathsf{st}_{\mathcal{E}_1}) \leftarrow \mathcal{E}_1(1^\kappa)$. It then runs Q honestly and lets it interact with G.
    * As part of the interaction, Q sends c to G
    * $\mathcal{E}^{\mathsf{Com}}_{\text{COMMIT}}$ extracts x as $x \leftarrow \mathcal{E}_2(\mathsf{st}_{\mathcal{E}_1}, c)$, samples a relaxed program $\Pi_x \in \mathring{\mathcal{P}}_{\text{COMMIT}}$ and outputs Π_x, $X^* = \{x\}$.
  - $\mathsf{Sim}^{\mathsf{Com}}_{\text{COMMIT}}\left(\pi_{c,\mathsf{crs}}, \widehat{\mu}^{(\cdot)}, \mathcal{I}_{\text{COMMIT}}, \Pi_x, \{x, \widehat{\mu}^{(\widehat{\beta})}(\text{REACH}_{\Pi_x}(x))\}\right)$:
    * The secrets available to the simulator are $\{\widehat{\beta}(2, i, x_i)\}_{i\in[l]}$.
    * Parse crs as $\{\mathsf{OT.crs}^i\}_{i\in[l]}$.
    * Parse c as $\{\mathsf{ots}_1^i\}_{i\in[l]}$.
    * For every $i \in [l]$, do the following:

$$\mathsf{ots}_2^i \leftarrow \begin{cases} \mathsf{OT}_2(\mathsf{OT.crs}^i, \mathsf{ots}_1^i, \widehat{\beta}(2, i, 0), \bot), & \text{if } x_i = 0, \\ \mathsf{OT}_2(\mathsf{OT.crs}^i, \mathsf{ots}_1^i, \bot, \widehat{\beta}(2, i, 1)), & \text{otherwise.} \end{cases}$$

    * Output the program $\rho[\mathsf{crs}, c, \{\mathsf{ots}_2^i\}_{i\in[l]}]$ (as described in Figure 19).

  The rest of the argument can be easily completed by considering a sequence of l hybrids[19] and using sender's security of the underlying OT scheme for proving indistinguishability between two consecutive hybrids.

□

B.2 Signature-Checking R3PO

In Appendix B.2.3, we prove Lemma 2 and in Appendix B.2.4 we prove Lemma 3.

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 $\mathsf{mpre} \preccurlyeq m$ to denote the predicate: mpre a prefix of m.

B.2.1 R3PO for One-time Signature-Checking

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

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

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

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

[19] To elaborate further on this, we consider a hybrid Hybrid^j ($j \in \{0, 1, \ldots, l\}$) where the first j ciphertexts $\{\mathsf{ots}_2^i\}_{i\in[l], i\le j}$ (in the description of the final program ρ) are generated as by $\mathsf{Sim}^{\mathsf{Com}}_{\text{COMMIT}}$, while the remaining ones are generated as in $\mathcal{O}^{\mathsf{Com}}_{\text{COMMIT}}$. 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 16.

R3PO scheme $\mathcal{O}^{\mathsf{Com}}_{\text{COMMIT}}$

Security Parameter: Let κ be the security parameter.

Let $\mathsf{OT} = (\mathsf{OT.Setup}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ be an OT scheme.

Let $\mathsf{Com} = (\mathsf{Com.Setup}, \mathsf{Com}, \mathsf{Com.Open})$ be a commitment opening scheme.

Input: Transition function $\pi_{c,\mathsf{crs}} \in \mathcal{P}^{\mathsf{Com}}_{\text{COMMIT}}$, message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{COMMIT}},1}$.

Obfuscator $\mathcal{O}^{\mathsf{Com}}_{\text{COMMIT}}\left(\pi_{c,\mathsf{crs}}, \widehat{\mu}^{(\widehat{\beta})}\right)$:

- Parse crs as $\{\mathsf{OT.crs}^i\}_{i\in[l]}$.
- Parse c as $\{\mathsf{ots}_1^i\}_{i\in[l]}$.
- For every $i \in [l]$, construct ciphertext $\mathsf{ots}_2^i \leftarrow \mathsf{OT}_2(\mathsf{OT.crs}^i, \mathsf{ots}_1^i, \widehat{\beta}(2,i,0), \widehat{\beta}(2,i,1))$.
- Output a program $\rho[\mathsf{crs}, c, \{\mathsf{ots}_2^i\}_{i\in[l]}]$ which does the following:
  - takes as input $d = \{\omega^i\}_{i\in[l]}$
  - parses crs as $\{\mathsf{OT.crs}^i\}_{i\in[l]}$
  - sets $x = \mathsf{Com.Open}(\mathsf{crs}, c, d)$
  - for every $i \in [l]$, sets $m_i = \mathsf{ots}_3(\mathsf{OT.crs}^i, \mathsf{ots}_2^i, x_i, \omega^i)$
  - outputs $\{m_i\}_{i\in[l]}$

Fig. 19. R3PO for Commitment-opening

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

R3PO. We now show that, for an adversary class $\mathcal{Q}^{\mathsf{1\text{-}Sig}}_{\text{1-sign}}$ with all PPT Q and a relaxed program class same as $\mathcal{P}^{\mathsf{1\text{-}Sig}}_{\text{1-sign}}$, a 1-Sig scheme and an R3PO for $\mathcal{G}^{\mathsf{1\text{-}Sig}}_{\text{1-sign}}$ can be constructed from any OTSE scheme. Recall that, the message output corresponding to a state σ_m in partition 2 is $\widehat{\mu}^{(\widehat{\beta})}(\sigma_m) = \{\widehat{\beta}(2, j, m_j)\}_{j\in|m|}$ (refer Definition 18).

An R3PO scheme for One-Time Signature Checking

Let κ be the security parameter.
Let OTSE be an OTSE scheme (OTSE.gen, OTSE.sign, OTSE.verify, OTSE.enc, OTSE.dec)
Let SKE be a symmetric-key encryption scheme (SKE.gen, SKE.enc, SKE.dec)

Obfuscator $\mathcal{O}_{1\text{-sign}}(\pi_{\mathsf{mpre}}[\sigma^1_{\mathsf{vk}}], \widehat{\mu}^{(\widehat{\beta})})$:

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

$$\mathsf{ct} \leftarrow \mathsf{SKE.enc}\big(k, \{\widehat{\beta}(2, j, \mathsf{mpre}_j\}_{j\in[t]}\big) \quad \Big\{\mathsf{ct}_j \leftarrow \mathsf{OTSE.enc}(\mathsf{vk}, j, \mathsf{mpre}_j, s_j)\Big\}_{j\in[t]}$$

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

$$x_{j,0} \leftarrow \mathsf{SKE.enc}(k, \widehat{\beta}(2, j, 0)) \qquad x_{j,1} \leftarrow \mathsf{SKE.enc}(k, \widehat{\beta}(2, j, 1))$$
$$\mathsf{ct}_{j,0} \leftarrow \mathsf{OTSE.enc}(\mathsf{vk}, j, 0, x_{j,0}) \quad \mathsf{ct}_{j,1} \leftarrow \mathsf{OTSE.enc}(\mathsf{vk}, j, 1, x_{j,1})$$

- output a program $\rho\left[\mathsf{vk}, \mathsf{ct}, \{\mathsf{ct}_j\}_{j\in[t]}, \{\mathsf{ct}_{j,b}\}_{j\in(t,n],b\in\{0,1\}}\right]$ [20], which does the following:
  - takes as input (τ, m).
  - decrypts each OTSE ciphertext as: $\forall j \in [t],\ x_j = \mathsf{OTSE.dec}(\mathsf{vk}, \tau, m, \mathsf{ct}_j)$
  - construct $k = x_1 + \ldots + x_t$
  - decrypts the SKE ciphertext to get labels: $\{\ell_j\}_{j\in[t]} = \mathsf{SKE.dec}(k, \mathsf{ct})$
  - decrypts each OTSE ciphertext as: $\forall j \in (t, n],\ y_j = \mathsf{OTSE.dec}(\mathsf{vk}, \tau, m, \mathsf{ct}_{j,m_j})$
  - decrypts the SKE ciphertexts to get labels: $\forall j \in (t, n],\ \ell_j = \mathsf{SKE.dec}(k, y_j)$
  - outputs $\{\ell_j\}_{j\in[n]}$.

Fig. 20. R3PO Obfuscation for one-time signature checking from OTSE.

Lemma 8. *If there exists a semi-honest* OTSE *scheme, then there exists a signature scheme* 1-Sig *and an R3PO scheme w.r.t.* $(\mathcal{G}^{\mathsf{1\text{-}Sig}}_{1\text{-}sign}, \mathcal{Q}^{\mathsf{1\text{-}Sig}}_{1\text{-}sign}, \mathring{\mathcal{P}}^{\mathsf{1\text{-}Sig}}_{1\text{-}sign})$.

Proof: We prove that the obfuscation scheme in Figure 20 is indeed a valid R3PO scheme w.r.t. $(\mathcal{G}^{\mathsf{1\text{-}Sig}}_{1\text{-sign}}, \mathcal{Q}^{\mathsf{1\text{-}Sig}}_{1\text{-sign}}, \mathring{\mathcal{P}}^{\mathsf{1\text{-}Sig}}_{1\text{-sign}})$. 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 $\langle \mathsf{H}^{\mathsf{1\text{-}Sig}}_{1\text{-sign}} \| Z : Q\rangle$ be $(\pi^{(\alpha)}, \widehat{\mu}^{(\widehat{\beta})}, a_G; a_Q)$. We now give an extractor $\mathcal{E}$ and simulator Sim for the scheme, so that it satisfies Definition 16:

- $\mathcal{E}$ trivially extracts the verification key vk^I from the last step of the interaction: where Q sends a target index I, message m from Q; and $\mathsf{H}^{\mathsf{1\text{-}Sig}}_{1\text{-sign}}$ replies with (vk^I, σ). $\mathcal{E}$ outputs $\pi[\mathsf{vk}^I]$ and $X^* = \{(m, \sigma)\}$.
- Sim takes as input $\pi^{(\cdot)}$, $\mu^{(\cdot)}$, Π and $\{\mu^{(\beta)}(\text{REACH}_\Pi(\overline{x}))\}_{\overline{x}\in X^*}$ (simply the labels $\{\widehat{\beta}(2, j, m_j)\}_{j\in[n]}$ for the message m). It outputs a simulated obfuscation $\widehat{\rho}$, where the OTSE ciphertexts corresponding to the bits $1 - m_j$ for each bit position $j \in (t, n]$ are encryptions of 0:

$$\widehat{\mathsf{ct}_{j,1-m_j}} \leftarrow \mathsf{OTSE.enc}(\mathsf{vk}, j, 1, \overline{0})$$

Indistinguishability follows via a reduction to the underlying OTSE security.

□

[20] where, ρ 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 $\mathsf{mpre} = m$, 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 use this key idea in Section 6.2.2 to give an R3PO for signature-checking where Q picks the verification key.

B.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 mpre, and the signature is now a sequence of one-time signatures along the path from root to mpre (where, each intermediate signature is on the verification keys of the child nodes). This allows us to have a single verification key vk^ϵ corresponding to the root node. Refer to Figure 21 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 $\Sigma^{\mathsf{1\text{-}Sig}}_{\mathrm{SIGN}^*} = \cup_{i=1}^{t}\big(\Sigma_{(i,0)} \cup \Sigma_{(i,1)}\big) \cup \Sigma_{t+1}$; where: for $i \in [t]$,

$$\Sigma_{(i,0)} = \{\sigma^{(i,0)}_{\mathsf{vk}} \mid \mathsf{vk} \in \mathcal{VK}\} \qquad \Sigma_{(i,1)} = \{\sigma^{(i,1)}_{\mathsf{vk}_0\|\mathsf{vk}_1} \mid \mathsf{vk}_0, \mathsf{vk}_1 \in \mathcal{VK}\}$$

and $\Sigma_{t+1} = \{\sigma^{t+1}_m \mid m \in \mathsf{M}\}$. The partition function is defined as $\mathcal{I}_{\mathrm{SIGN}^*}(\sigma^{(i,b)}_x) = 2 * (i-1) + b$, for $\sigma^{(i,b)}_x \in \Sigma_{(i,b)}$.

Each program in $\mathcal{P}^{\mathsf{1\text{-}Sig}}_{\mathrm{SIGN}^*}$, with start state $\sigma^{(1,0)}_{\mathsf{vk}^\epsilon} \in \Sigma_{(1,0)}$, is parameterized by a fixed prefix mpre and is defined by the transition function:

$$\begin{aligned}
\pi_{\mathsf{mpre}}(\sigma^{(i,0)}_{\mathsf{vk}}, y, \tau) &= \sigma^{(i,1)}_y && \text{if } i \neq t \text{ and } \mathsf{1\text{-}Sig.verify}(\mathsf{vk}, y, \tau) = 1\\
\pi_{\mathsf{mpre}}(\sigma^{(i,1)}_{\mathsf{vk}_0||\mathsf{vk}_1}, \epsilon) &= \sigma^{(i+1,0)}_{\mathsf{vk}_b} && \text{where } b = \mathsf{vk}_{\mathsf{mpre}_i}\\
\pi_{\mathsf{mpre}}(\sigma^{(t,0)}_{\mathsf{vk}}, m, \tau) &= \sigma^{t+1}_m && \text{if } \mathsf{mpre} = (m)_i \text{ and } \mathsf{1\text{-}Sig.verify}(\mathsf{vk}, m, \tau) = 1
\end{aligned}$$

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

Notation. We denote sk^i, vk^i to denote signature key pair sampled using randomness $\mathsf{F.eval}(s, i)$. We define $\mathsf{sign}^*(s, m)$ as a sequence of signatures along the path via $(m)_t$, i.e.. $\Big(\{\mathsf{vk}^i, \tau_i\}, \mathsf{vk}^{(m)_t}, \tau\Big)$ for $i \in [t]$, where each $\tau_i = \mathsf{1\text{-}Sig.sign}(\mathsf{sk}^i, \mathsf{vk}^{i0}||\mathsf{vk}^{i1})$ and $\tau = \mathsf{1\text{-}Sig.sign}(\mathsf{sk}^{(m)_t}, m)$.

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

- It samples a PRF key $s \to \mathsf{F.gen}(1^\kappa)$.
- It sends the root verification key vk^ϵ to Q.

– It accepts polynomially many queries of messages $m^j \in \mathsf{M}$ from Q and responds with $\mathsf{sign}^*(s, m^j)$.
– It then accepts a target message m from Q, verifies that Q had not queried any message with prefix $\mathsf{mpre} = (m)_t$ and punctures the PRF key on each prefix pre of mpre as $\mathsf{PRF.punct}(s, \mathsf{pre})$.
– It finally accepts a message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{SIGN}^*},0}$ from Z and outputs $(\pi_{\mathsf{mpre}}[\sigma^1_{\mathsf{vk}^\epsilon}], \widehat{\mu}^{(\widehat{\beta})}, s)$.

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

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

$$\mathcal{L} = \left(\mathcal{L}_{(1,0)}, \mathcal{L}_{(1,1)}, \ldots, \mathcal{L}_{(t,0)}\right),$$

where

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

We prove this by explicitly giving a J, W, Z that satisfy Definition 17.

– Case $b = 0$: At this partition, $\mathcal{G}^{\text{1-Sig}}_{\text{SIGN}^*}$ decomposes to $\mathcal{G}^{\text{1-Sig}}_{\text{1-sign}}$ (with the appropriate prefix and partition function, that we skip here; refer Section 6.5). Let $G \in \mathcal{G}^{\text{1-Sig}}_{\text{SIGN}^*}$ be a generator that interacts with $Q \in \mathcal{Q}$.
  - **Defining** W: W runs Q in a white-box way and interacts with either J or $\mathsf{H}^{\text{1-Sig}}_{\text{1-sign}}$ via the bottom channel.
    * W sends query $k = \epsilon$ on the bottom channel and gets back vk^ϵ.
    * For each query m^j that Q makes in the top channel, it does the following. For each $k \prec \mathsf{mpre}^j$ [21], it queries indices $k||0$, $k||1$ on the bottom channel and gets back vk^{k0}, vk^{k1}; it then queries $(k, \mathsf{vk}^{k0}||\mathsf{vk}^{k1})$ on the bottom channel and gets back τ^j_k. It finally queries (mpre^j, m^j) on the bottom channel and gets back τ^j_{k+1}.
    * Similarly, for the target message m, prefix mpre.
    * W finally sends the target index $I = (m)_i$, target message $\mathsf{vk}^{k0}||\mathsf{vk}^{k1}$ 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 vk^ϵ on the top channel. W sends query ϵ on the bottom channel. J sends vk^ϵ on the bottom channel.
    * For each query m^j that Q makes and G responds with τ^j on the top channel, it does the following. It responds to the queries of W from τ^j.
    * Similarly, for the target message m, prefix mpre.
    * It finally samples $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{SIGN}^*},i}$, receives target index I, target message m and prefix size x; sets target prefix mpre as $(m)_x$ and halts with output $\widehat{\pi}_{\mathsf{mpre}}[\sigma^{(i,0)}_{\mathsf{vk}^I}]$, $\widehat{\mu}^{(\widehat{\beta})}$).
  - **Defining** Z**:** Z runs $\mathsf{H}^{\text{1-Sig}}_{\text{1-sign}}$ in a black-box way and interacts with Q via the top channel.
    * W queries ϵ in the bottom channel and $\mathsf{H}^{\text{1-Sig}}_{\text{1-sign}}$ responds with vk^ϵ. Z sends vk^ϵ 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 $\mathsf{H}^{\text{1-Sig}}_{\text{1-sign}}$ 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.

[21] that is, all prefixes k of m^j upto length $t - 1$

* Z finally samples $\mu^{(\beta)} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{SIGN}^*},0}$ and halts with output $(\pi^{(\alpha)}, \mu^{(\beta)})$.

– Case $b = 1$: At this partition, $\mathcal{G}^{\text{1-Sig}}_{\text{SIGN}^*}$ decomposes to $\mathcal{G}^{\Sigma}_{\epsilon}$ (with the appropriate prefix and partition function, that we skip here; refer Section 6.5). Let $G \in \mathcal{G}^{\text{1-Sig}}_{\text{SIGN}^*}$ be a generator that interacts with $Q \in \mathcal{Q}$.

- **Defining** W: W runs Q in a white-box way and interacts with either J or $\mathsf{H}^{\Sigma}_{\epsilon}$ 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 τ be the response. Let I be the $i-1$-bit prefix of m and b be m_i; W extracts the keys vk^{I0}, vk^{I1} from τ and sends the states $\sigma_1 = \sigma^{(i,1)}_{\mathsf{vk}^{I0}||\mathsf{vk}^{I1}}$, $\sigma_2 = \sigma^{(i+1,0)}_{\mathsf{vk}^{Ib}}$ 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 $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{SIGN}^*},i}$, receives receives the states σ_1, σ_2 from W and halts with output $\pi_{\sigma_2}[\sigma_1]$, $\widehat{\mu}^{(\widehat{\beta})}$).
- **Defining** Z**:** Z runs $\mathsf{H}^{\Sigma}_{\epsilon}$ 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 9 and Theorem 1.

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

B.2.3 R3PO for a Puncturable Signature Scheme

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

Lemma 10. *If there exists an one-time signature primitive* 1-Sig *with an R3PO scheme w.r.t.* $\mathcal{G}^{\text{1-Sig}}_{SIGN^*}$*, then there exists a puncturable signature scheme* Sig *and an R3PO scheme w.r.t.* $(\mathcal{G}^{\text{Sig},\mathcal{M}}_{SIGN}, \mathcal{Q}, \mathring{\mathcal{P}}^{\text{Sig}}_{SIGN})$ *for any message function space* $\mathcal{M}$*.*

Proof: Sig corresponds to the signature scheme that was implicit in the generator $\mathsf{H}^{\text{1-Sig}}_{\text{SIGN}^*}$ above. We explicitly state it again in Figure 21. In the figure, the symbol $(x)_i$ denotes the i-bit long prefix of the string x. We describe the wrapper on the R3PO of $\mathcal{G}^{\text{1-Sig}}_{\text{SIGN}^*}$ in Figure 22.

□

We now prove the main lemma.

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

Proof: This result follows from Lemma 8, Corollary 1 and Lemma 10. We prove this for the message function class $\widehat{\mathcal{M}}$ (Definition 18). 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 $\widehat{\mathcal{M}}$.

To build the required R3PO, we first build an R3PO for a one-time signature-checking scheme (Appendix B.2.1). We then build an R3PO for a signature-checking scheme, but with multiple transitions (Appendix B.2.2). We finally squish this to get the required construction (Appendix B.2.3). □

B.2.4 R3PO for Signature Checking with Adversarial Key

Lemma 3 (Restated)**.** *If there exists a semi-honest secure OTSE scheme, then there exists a puncturable signature scheme* Sig *and a PPT program* $\mathcal{O}_{\mathsf{Sig}}$ *s.t.* $\mathcal{O}_{\mathsf{Sig}}$ *is an R3PO obfuscation scheme w.r.t.* $(\mathcal{G}^{\text{Sig}}_{QSIGN}, \mathcal{Q}, \mathring{\mathcal{P}}^{\text{Sig}}_{QSIGN})$ *as well as* $(\mathcal{G}^{\text{Sig}}_{SIGN\text{-}PO}, \mathcal{Q}, \mathring{\mathcal{P}}^{\text{Sig}}_{SIGN\text{-}PO})$*.*

Security Parameter: Let κ be the security parameter.

Let $\mathsf{1\text{-}Sig} = (\mathsf{1\text{-}Sig.gen}, \mathsf{1\text{-}Sig.sign}, \mathsf{1\text{-}Sig.verify})$ be a one-time signature scheme.

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

- $\mathsf{Sig.gen}(1^\kappa) \to (\mathsf{Sig.vk}, \mathsf{Sig.sk})$:

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

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

- $\mathsf{Sig.sign}(\mathsf{Sig.sk}, x) \to \tau$: Parse $\mathsf{Sig.sk}$ as s.

$$(\mathsf{1\text{-}Sig.vk}^\epsilon, \mathsf{1\text{-}Sig.sk}^\epsilon) \leftarrow \mathsf{1\text{-}Sig.gen}(1^\kappa; \mathsf{F.eval}(s, \epsilon)).$$
$$\forall i \in [t]:$$
$$(\mathsf{1\text{-}Sig.vk}_0^{i+1}, \mathsf{1\text{-}Sig.sk}_0^{i+1}) \leftarrow \mathsf{1\text{-}Sig.gen}(1^\kappa; \mathsf{F.eval}(s, (x)_i||0)),$$
$$(\mathsf{1\text{-}Sig.vk}_1^{i+1}, \mathsf{1\text{-}Sig.sk}_1^{i+1}) \leftarrow \mathsf{1\text{-}Sig.gen}(1^\kappa; \mathsf{F.eval}(s, (x)_i||1)),$$
$$m_i = \mathsf{1\text{-}Sig.vk}_0^{i+1}||\mathsf{1\text{-}Sig.vk}_1^{i+1},$$
$$\tau_i \leftarrow \mathsf{1\text{-}Sig.sign}(\mathsf{1\text{-}Sig.sk}_{x_i}^i, m_i).$$

  Finally, do

$$\tau_x \leftarrow \mathsf{1\text{-}Sig.sign}(\mathsf{1\text{-}Sig.sk}_{x_t}^t, x).$$

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

- $\mathsf{Sig.verify}(\mathsf{Sig.vk}, \tau, x) \to b$: Parse τ as $\Big(\{(\tau_i, m_i)\}_{i\in[t]},\ (\tau_x, x)\Big)$.

$$\forall i \in [t]:$$
$$\text{Parse } m_i \text{ as } \mathsf{1\text{-}Sig.vk}_0^{i+1}||\mathsf{1\text{-}Sig.vk}_1^{i+1},$$
$$b_i = \mathsf{1\text{-}Sig.verify}(\mathsf{1\text{-}Sig.vk}_{x_b}^i, \tau_i, m_i).$$

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

- $\mathsf{Sig.punct}(\mathsf{Sig.sk}, \mathsf{mpre}) \to \mathsf{Sig.sk}'$: Output $\mathsf{F.punct}(s, \mathsf{mpre})$.

- $\mathsf{Sig.psign}(\mathsf{Sig.sk}', x) \to \tau'$: Output $\mathsf{Sig.sign}(\mathsf{Sig.sk}', x)$.

Fig. 21. Puncturable Signature from One-Time Signature

Proof: Note that, in the construction of R3PO for $\mathcal{G}^{\mathsf{1\text{-}Sig}}_{\text{SIGN}^*}$ from $\mathcal{G}^{\mathsf{1\text{-}Sig}}_{\text{1-sign}}$ (Appendix B.2.2), we had a target prefix mpre in the reactive program, and the reach-restriction to prefix mpre was enforced via epsilon-transitions (where it transitions along the path to mpre) and an explict prefix check in the last one-step transition was not needed. The intention of decomposing at the last transition to $\mathcal{G}^{\mathsf{1\text{-}Sig}}_{\text{1-sign}}$ with a target prefix mpre was to allow the same obfuscation to also be a valid R3PO for $\mathcal{G}^{\mathsf{Sig}}_{\text{QSIGN}}$. This follows from the fact that the R3PO for $\mathcal{G}^{\mathsf{1\text{-}Sig}}_{\text{1-sign}}$ 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.

We define $\mathcal{G}^{\mathsf{Sig}}_{\text{SIGN-PO}}$ as a set of generators in $\mathcal{G}^{\mathsf{Sig}}_{\text{SIGN}}$ that output reactive programs with full message as the target prefix, that is, $|\mathsf{mpre}| = n$. Such a generator decomposes as in Lemma 9; where in, it decomposes

R3PO scheme $\mathcal{O}^{\mathsf{Sig}}_{\text{SIGN}}$

$\mathcal{O}^{\mathsf{Sig}}_{\text{SIGN}}(\pi_{\mathsf{mpre}}[\sigma^1_{\mathsf{vk}}], \widehat{\mu}^{(\widehat{\beta})})$:

- Parse vk as $\mathsf{1\text{-}Sig.vk}^{\epsilon}$.
- Sample $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{SIGN}^*},0}$.
- Set $\rho \leftarrow \mathcal{O}^{\mathsf{1\text{-}Sig}}_{\text{SIGN}^*}(\pi^*_{\mathsf{mpre}}[\sigma^1_{\mathsf{1\text{-}Sig.vk}^{\epsilon}}], \widehat{\mu}^{(\widehat{\beta})})$.
- Outputs a program $\rho'[\rho]$ as follows:
  - Takes as input (τ, x), where τ is parsed as $\Big(\{\tau_i\}_{i\in[t]},\ (\tau_x, x)\Big)$.
  - Traverse the program ρ by sequentially providing input τ_i interleaved with ϵ-transitions, for every $i \in [t]$.
  - Finally input (τ_x, x) and output ρ's outputs.

Fig. 22. Signature checking obfuscator

to $\mathcal{G}^{\mathsf{1\text{-}Sig}}_{\text{1-sign}}$ at each partition $(i, 0)$ with target prefix length 0 (that is, no target prefix) for each $i < n$ and decomposes to $\mathcal{G}^{\mathsf{1\text{-}Sig}}_{\text{1-sign}}$ 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 21), the R3PO for $\mathcal{G}^{\mathsf{Sig}}_{\text{SIGN-PO}}$ is also a valid R3PO for $\mathcal{G}^{\mathsf{Sig}}_{\text{QSIGN}}$ over the same relaxed program class. Formally,

- Extractor $\mathcal{E}$: For an adversary Q, it trivially extracts the verification key vk and target message m from the transcript of the interaction between any $G \in \mathcal{G}^{\mathsf{Sig}}_{\text{QSIGN}}$ and Q. It extracts the signature signing key sk from Q, constructs the signature for m and outputs the same relaxed program as in $\mathcal{G}^{\mathsf{Sig}}_{\text{SIGN-PO}}$.
- Simulator Sim: it takes as input $\pi^{(\cdot)}$, $\mu^{(\cdot)}$, Π_m and $\{\mu^{(\beta)}(\text{REACH}_{\Pi}(\overline{x}))\}_{\overline{x}\in X^*}$ (simply the labels for m). It outputs a simulated obfuscation $\widehat{\rho}$ as follows. It internally runs the obfuscator for $\mathcal{G}^{\mathsf{Sig}}_{\text{SIGN-PO}}$ and outputs its output.

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

□

B.3 Hash-Opening 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 B.3.1). We then do domain extension to get a hash scheme that supports arbitrary compression and give a R3PO wr.t. this hash scheme, but for reactive programs with multiple transitions (Appendix B.3.2). We finally squish it to get a one-step transition reactive program class and R3PO for it (Appendix B.3.3).

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

We now show how to construct a hash scheme $\mathsf{2\text{-}Hash}$ with factor-2 compression [22] and an R3PO scheme for hash opening instantiated with this scheme (that is, for $(\mathcal{G}^{\mathsf{2\text{-}Hash}}_{\text{HASH}}, \mathcal{Q}^{\mathsf{2\text{-}Hash}}_{\text{HASH}}, \mathring{\mathcal{P}}^{\mathsf{2\text{-}Hash}}_{\text{HASH}})$) from laconic OT with factor-2 compression.

[22] where, an input of two blocks is hashed to a digest of one block and the block index for hash opening is simply a bit

Block-Openable CRH with Factor-2 Compression

Let $\ell OT = (\mathsf{crsGen}, \mathsf{hash}, \mathsf{Send}, \mathsf{Receive})$ be a laconic OT with factor-2 compression.

Scheme 2-Hash:

- $\mathsf{crsGen}(1^\kappa)$: Compute $\mathsf{crs} \leftarrow \ell OT.\mathsf{crsGen}(1^\kappa)$ and output crs.
- $\mathsf{hash}(\mathsf{crs}, m)$: Compute $\mathsf{digest} := \ell OT.\mathsf{hash}(\mathsf{crs}, m)$ and output digest.
- $\mathsf{openBlock}(\mathsf{crs}, m, b)$: Output $w := m$.
- $\mathsf{acceptBlock}(\mathsf{crs}, \mathsf{digest}, b, w)$: Parse w as $m = m_0 \| m_1$ such that $|m_0| = |m_1|$. If $\ell OT.\mathsf{hash}(\mathsf{crs}, m) = \mathsf{digest}$, then output m_b, otherwise output $\perp$.

Fig. 23. 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.*

Proof: We show that the scheme in Figure 23 is a block-openable CRH with factor-2 compression.

- Correctness: from the correctness of the underlying ℓOT scheme, hash must be deterministic. Thus, $\mathsf{acceptBlock}$ outputs correctly with probability 1.
- Factor-2 Compression: reduces to the compression of the underlying ℓOT scheme.
- Collision Resistance: we reduce this to the sender privacy of the underlying ℓOT scheme. Let Adv be an adversary that outputs (m, b, w) with probability α s.t. $m := m_0 || m_1$ and $\mathsf{acceptBlock}(\mathsf{crs}, \mathsf{hash}(\mathsf{crs}, m), b, w) \notin \{\perp, m_b\}$. We build an adversary Adv', distinguisher $\mathcal{D}'$ for the sender privacy experiment of ℓOT as follows:
  - Adv' internally runs Adv in straight-line blackbox way and behaves as follows. It receives crs from the experiment and forwards it to Adv. It receives (m, b, w) from Adv. Let $w := w_0 || w_1$. If $m_b \neq w_b$, it chooses an index L s.t. $m[L] \neq w[L]$ [23]; if $m_b = w_b$, then it chooses a random L. It sets $D = m$, samples challenge messages x_0, x_1 s.t. $x_0 \neq x_1$ and sets $aux = (\mathsf{crs}, m, w, L, x_0, x_1)$; it finally outputs (aux, x_0, x_1, D, L) to the ℓOT experiment.
  - $\mathcal{D}'$ gets $(aux = (\mathsf{crs}, m, w, L, x_0, x_1), \mathsf{ct})$ from the ℓOT experiment and computes the following:

$$x' := \mathsf{Receive}^m(\mathsf{crs}, \mathsf{ct}, L) \qquad x'' := \mathsf{Receive}^w(\mathsf{crs}, \mathsf{ct}, L).$$

  If $x' \neq x''$, it outputs 0. Otherwise, it outputs 1.

  Now, $\mathcal{D}'$ is able to distinguish between the real world and ideal world if Adv outputs a valid collision (that is, $m[L] \neq w[L]$) and the ℓOT simulator is unable to guess the other message, in which case $x' = x_{m[L]}$ and $x'' = x_{1-m[L]}$. But, from the security of ℓOT, its advantage must be negligible. Thus, α must also be negligible.

□

[23] where, $x[L]$ represents the bit at index position L in x

An R3PO scheme for 2-Hash

Let κ be the security parameter.
Let $\ell OT = (\mathsf{crsGen}, \mathsf{hash}, \mathsf{Send}, \mathsf{Receive})$ be a laconic OT with factor-2 compression.
Let $\text{2-Hash} = (\mathsf{crsGen}, \mathsf{hash}, \mathsf{openBlock}, \mathsf{acceptBlock})$ be as in Figure 23.

Obfuscator $\mathcal{O}^{\text{2-Hash}}_{\text{HASH}}(\pi_{\mathsf{digest},b,\mathsf{crs}}, \widehat{\mu}^{(\widehat{\beta})})$:

- Let $t = |\mathsf{digest}|$.
- If $b = 0$, then $t_0 := 0$ else $t_0 := t$.
- For each bit position $j \in [t]$, construct a single ℓOT ciphertext that encrypts the labels $\widehat{\beta}(2, j, \cdot)$ for bit position j as follows
$$\mathsf{ct}_j \leftarrow \ell OT.\mathsf{Send}(\mathsf{crs}, \mathsf{digest}, j + t_0, \widehat{\beta}(2, j, 0), \widehat{\beta}(2, j, 1)).$$
- Output $\rho\left[\mathsf{crs}, b, \{\mathsf{ct}_j\}_{j\in[t]}\right]$.

Evaluating an obfuscation $\rho\left[\mathsf{crs}, b, \{\mathsf{ct}_j\}_{j\in[t]}\right]$ **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 ℓOT ciphertext as follows:
$$m_j := \ell OT.\mathsf{Receive}^D(\mathsf{crs}, \mathsf{ct}_j, j + t_0).$$
- Output $\{m_j\}_{j\in[t]}$.

Fig. 24. R3PO for Hash Opening for block-openable CRH with factor-2 compression.

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

Proof: We instantiate 2-Hash with the scheme in Figure 23. We now show that the scheme in Figure 24 is a R3PO scheme w.r.t. $(\mathcal{G}^{\text{2-Hash}}_{\text{HASH}}, \mathcal{Q}^{\text{2-Hash}}_{\text{HASH}}, \mathring{\mathcal{P}}^{\text{2-Hash}}_{\text{HASH}})$. Correctness follows from the correctness of the underlying ℓOT scheme. For proving security, we will have to describe extractor $\mathcal{E}$ and simulator Sim for the scheme, so that it satisfies Definition 16. Let Z, Q be arbitrary parties s.t. the interaction between $\mathsf{H}^{\text{2-Hash}}_{\text{HASH}}, Z$ and Q follows the description of generator class $\mathcal{G}^{\text{2-Hash}}_{\text{HASH}}$ in Section 6.3. 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 $(\pi_{\mathsf{digest},b,\mathsf{crs}}, X^* = \{D\})$, where (digest, b) is sent by Q to $\mathsf{H}^{\mathsf{Hash}}_{\text{HASH}}$ in the interaction. The simulator Sim behaves as follows:

1. Takes as input $\left(\pi_{\mathsf{digest},b,\mathsf{crs}}, \widehat{\mu}^{(\cdot)}, \pi_{\mathsf{digest},b,\mathsf{crs}}, \{x, \widehat{\mu}^{(\widehat{\beta})}(\text{REACH}_{\pi_{\mathsf{digest},b,\mathsf{crs}}}(x))\}_{x\in X^*}\right)$.
2. Parse $x = D$ in X^*.
3. Let $t = |\mathsf{digest}|$.
4. If $b = 0$, then $t_0 := 0$ else $t_0 := t$.
5. For each bit position $j \in [t]$, construct a single ℓOT ciphertext as follows
$$\mathsf{ct}_j \leftarrow \ell OT.\mathsf{Sim}(\mathsf{crs}, D, j + t_0, \widehat{\beta}(2, j, D[j + t_0])).$$
6. Output $\rho\left[\mathsf{crs}, b, \{\mathsf{ct}_j\}_{j\in[t]}\right]$.

Indistinguishability between the distributions $\text{REAL}(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}, Q)$ and $\text{IDEAL}(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}, Q)$ can be shown by a sequence of t hybrids, where $t = |D|/2 = |\mathsf{digest}|$. In the i^{th} hybrid for every $i \in [t]$, we change how the

ciphertext ct_i is being generated inside the output program $\rho[\mathsf{crs}, b, \{\mathsf{ct}_j\}_{j\in[t]}]$. While in the $(i-1)^{\text{th}}$ hybrid ct_i is obtained by the output of $\ell OT.\mathsf{Send}$ with the other message being $\widehat{\beta}(2, i, 1-D[i+t_0])$, in the i^{th} hybrid it is chosen to be the output of $\ell OT.\mathsf{Sim}$ with only $\widehat{\beta}(2, i, D[i+t_0])$ provided. Proving indistinguishability between these 2 hybrids reduces to sender security of ℓOT. Note that the first hybrid is the same as REAL while the final hybrid is same as IDEAL. □

B.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 for block-openable CRH with factor-2 compression.

Program Family and Generator Class. The generator class is same as $\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}$ described at the beginning of Section 6.3. At the end of the interaction, a generator in the class outputs a reactive program in $\mathcal{P}^{\mathsf{Hash}}_{\text{HASH}^*}$ parameterized by a hash digest, a location L (specified as a sequence of bits $(b_1, b_2, \ldots, b_l)$), a common random string crs; with start state $\sigma^1_{\mathsf{digest}}$ and transition function w.r.t. a state space $\Sigma_{\text{HASH}^*} = \{\sigma^i_y \mid i \in [l+1], y \in \{0,1\}^\kappa\}$ as follows:

$$\pi_{\mathsf{digest},L,\mathsf{crs}}(\sigma^i_x, w) = \begin{cases} \sigma^{i+1}_y & \text{if } \mathsf{acceptBlock}(\mathsf{crs}, x, b_i, w) = y. \end{cases}$$

The partition function is defined as $\mathcal{I}_{\text{HASH}^*}(\sigma^i_y) = i$ for all $y \in \{0,1\}^\kappa$.

R3PO. We now show that there is an R3PO scheme w.r.t. $(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}^*}, \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}^*}, \mathring{\mathcal{P}}^{\mathsf{Hash}}_{\text{HASH}^*})$ by decomposing $(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}^*}, \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}^*})$ into a library of hash openings $(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}, \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}})$ and then invoking Theorem 1.

Lemma 13. *$(\mathcal{G}^{\mathsf{Hash}}_{HASH^*}, \mathcal{Q}^{\mathsf{Hash}}_{HASH^*})$ decomposes into a library of R3PO $(\mathcal{G}^{\mathsf{Hash}}_{HASH}, \mathcal{Q}^{\mathsf{Hash}}_{HASH})$.*

Proof: We will show that $(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}^*}, \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}^*})$ decomposes into the library $\mathcal{L} = (\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}, \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}}, \mathring{\mathcal{P}}^{\mathsf{Hash}}_{\text{HASH}})_{i\in[l]}$. Therefore, for a generator $G \in \mathcal{G}^{\mathsf{Hash}}_{\text{HASH}^*}$ and a part $i \in [l]$, we will describe J, Z, W so that $\forall Q \in \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}^*}$, and all (i)-partial reach-extractors $\mathcal{E}$ for Q, the conditions given in Definition 17 hold.

- **Defining J.** Here is a description of J :
  1. Reads crs from $\mathsf{H}^{\mathsf{Hash}}_{\text{HASH}^*}$.
  2. Reads (digest', b) from W.
  3. Sends a freshly sampled message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{HASH}^*},1}$ to $\mathsf{H}^{\mathsf{Hash}}_{\text{HASH}^*}$.
  4. Outputs the program $\pi_{\mathsf{digest}',b,\mathsf{crs}} \in \mathcal{P}^{\mathsf{Hash}}_{\text{HASH}}$.
- **Defining Z.** Here is a description of Z :
  1. Reads crs from $\mathsf{H}^{\mathsf{Hash}}_{\text{HASH}}$.
  2. Reads (digest, L) from Q.
  3. Also sends a freshly sampled message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{HASH}},1}$ to $\mathsf{H}^{\mathsf{Hash}}_{\text{HASH}}$.
  4. Outputs the program $\pi_{\mathsf{digest},L,\mathsf{crs}} \in \mathcal{P}^{\mathsf{Hash}}_{\text{HASH}^*}$.
- **Defining W.** It reads the database D and location L from Q. It then computes the node digest' and bit b based on the i^{th}-level in the Merkle tree with D representing the leaves. The bit b represents the corresponding bit in the binary representation for the location L. It sends (digest', b) to both J and $\mathsf{H}^{\mathsf{Hash}}_{\text{HASH}}$.

□

The following follows immediately from Lemma 13 and Theorem 1.

Corollary 2. *If there exists a block-openable CRH with factor-2 compression Hash, then there exists an R3PO scheme w.r.t. $(\mathcal{G}^{\mathsf{Hash}}_{HASH^*}, \mathcal{Q}^{\mathsf{Hash}}_{HASH^*}, \mathring{\mathcal{P}}^{\mathsf{Hash}}_{HASH^*})$.*

B.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 $\mathcal{P}^{\mathsf{Hash}}_{\text{HASH}^*}$ above.

Lemma 14. *If there exists a block-openable CRH with factor-2 compression* Hash'*, then exists a block-openable CRH with arbitrary compression* Hash*.*

Proof: The description of Hash from Hash' has been given in Figure 25. For ease of exposition, assume that $n = 2^d$ for some $d \in \mathbb{Z}$, and input messages consist of n blocks of κ length each. While Hash can be instantiated using a Merkle tree with Hash' being the underlying hash scheme, $\mathsf{openBlock}$ 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. $\mathsf{acceptBlock}$ 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 Hash' at the node where the two sequences diverge. We skip mentioning the full details here as this is a standard construction.

Let $\mathsf{Hash}' = (\mathsf{crsGen}, \mathsf{hash}, \mathsf{openBlock}, \mathsf{acceptBlock})$ be a doubly-compressing hash scheme.

- $\mathsf{crsGen}(1^\kappa)$: Run $\mathsf{crs} \leftarrow \mathsf{Hash}'.\mathsf{crsGen}(1^\kappa)$ and output crs.
- $\mathsf{hash}(\mathsf{crs}, m)$: Let $m = m_1 \| \ldots \| m_n$ s.t. $|m_i| = \kappa$ for all $i \in [n]$. Construct a Merkle tree with $m_1, \ldots, m_n$ representing the leaves and $\mathsf{Hash}'.\mathsf{hash}(\mathsf{crs}, \cdot)$ being the doubly-compressing hash function i.e., in any iteration, club two consecutive nodes on the same level and compute the hash on this pair. Repeat this process until a single string digest of length κ remains at the root. Output digest.
- $\mathsf{openBlock}(\mathsf{crs}, m, i)$: 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.
- $\mathsf{acceptBlock}(\mathsf{crs}, \mathsf{digest}, i, k)$: Parse k as $k_1, \ldots, k_d$, where d is the depth of the Merkle tree. Check if $k_{j,b_j} = \mathsf{Hash}'.\mathsf{hash}(\mathsf{crs}, k_{j+1})$, for all $j \in [d]$, where b_j denotes the child chosen at depth j to reach the i^{th} leaf.

Fig. 25. Arbitrarily-Compressing Hash from Doubly-Compressing Hash

□

Lemma 4 (Restated)**.** *If there exists a laconic OT scheme with factor-2 compression, then there exists an arbitrarily-compressing hash scheme* Hash *and an R3PO scheme w.r.t.* $(\mathcal{G}^{\mathsf{Hash}}_{HASH}, \mathcal{Q}^{\mathsf{Hash}}_{HASH}, \mathring{\mathcal{P}}^{\mathsf{Hash}}_{HASH})$*.*

Proof: We instantiate the block-openable CRH with arbitrary compression Hash with the scheme in Figure 25. To get an R3PO scheme $\mathcal{O}^{\mathsf{Hash}}_{\text{HASH}}$ w.r.t. $(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}, \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}}, \mathring{\mathcal{P}}^{\mathsf{Hash}}_{\text{HASH}})$, we must put a wrapper around the R3PO scheme $\mathcal{O}^{\mathsf{Hash}'}_{\text{HASH}^*}$ w.r.t. $(\mathcal{G}^{\mathsf{Hash}'}_{\text{HASH}^*}, \mathcal{Q}^{\mathsf{Hash}'}_{\text{HASH}^*}, \mathcal{P}^{\mathsf{Hash}'}_{\text{HASH}^*})$, as obtained in Corollary 2. Consider the description given in Figure 26. Then we prove the following properties:

- **Correctness.** This follows from the description of Hash, sampling of $\widehat{\beta}$ and correctness of $\mathcal{O}^{\mathsf{Hash}'}_{\text{HASH}^*}$.
- **Efficiency.** This follows from the efficiency of $\mathcal{O}^{\mathsf{Hash}'}_{\text{HASH}^*}$.
- **Security.** We need to provide a $\mathsf{Sim}^{\mathsf{Hash}}_{\text{HASH}}$ s.t. $\forall\, G \in \mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}$ and $\forall Q \in \mathcal{Q}^{\mathsf{Hash}}_{\text{HASH}}$, there exists a reach-extractor $\mathcal{E}^{\mathsf{Hash}}_{\text{HASH}}$ w.r.t. $(\mathcal{G}^{\mathsf{Hash}}_{\text{HASH}}, \mathring{\mathcal{P}}^{\mathsf{Hash}}_{\text{HASH}})$ which satisfies Definition 16. We describe these algorithms as follows:
  - $\mathcal{E}^{\mathsf{Hash}}_{\text{HASH}}$:
    - Since the interaction between G and Q is the same as that in the HASH^* generator class, $\mathcal{E}^{\mathsf{Hash}}_{\text{HASH}}$ could just invoke that extractor $\mathcal{E}^{\mathsf{Hash}'}_{\text{HASH}^*}$ 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 $\pi_{\mathsf{digest},i,\mathsf{crs}} \in \mathcal{P}^{\mathsf{Hash}}_{\text{HASH}}$, where $(\mathsf{digest}, i, \mathsf{crs})$ are obtained from the interaction.
  - $\mathsf{Sim}^{\mathsf{Hash}}_{\text{HASH}}\left(\pi_{\mathsf{digest},i,\mathsf{crs}}, \widehat{\mu}^{(\cdot)}, \pi_{\mathsf{digest},i,\mathsf{crs}}, \{x, \widehat{\mu}^{(\widehat{\beta})}(\text{REACH}_{\pi_{\mathsf{digest},i,\mathsf{crs}}}(x))\}_{x \in X^*}\right)$:
    - Parse $x = k = (k_1, \ldots, k_d)$, and set X as the sequence k only.
    - Obtain $\rho \leftarrow \mathsf{Sim}^{\mathsf{Hash}'}_{\text{HASH}^*}(\pi'_{\mathsf{digest},i,\mathsf{crs}}, \widehat{\mu}^{(\cdot)}, \pi'_{\mathsf{digest},i,\mathsf{crs}}, \{x, \widehat{\mu}^{(\widehat{\beta}')}(\text{REACH}_{\pi'_{\mathsf{digest},i,\mathsf{crs}}}(x))\}_{x \in X})$, where $\pi'_{\mathsf{digest},i,\mathsf{crs}} \in \mathcal{P}^{\mathsf{Hash}'}_{\text{HASH}^*}$ and $\widehat{\beta}'$ is generated from $\widehat{\beta}$ as in Figure 26.

R3PO scheme $\mathcal{O}^{\mathsf{Hash}}_{\mathtt{HASH}}$

$\mathcal{O}^{\mathsf{Hash}}_{\mathtt{HASH}}(\pi_{\mathsf{digest},i,\mathsf{crs}}, \widehat{\mu}^{(\widehat{\beta})})$:

- Consider the program $\pi'_{\mathsf{digest},i,\mathsf{crs}} \in \mathcal{P}^{\mathsf{Hash}'}_{\mathtt{HASH}^*}$ as described in Appendix B.3.2.
- Consider the new message function $\widehat{\beta}'$ as follows:

$$\widehat{\beta}'(i', j, b) = \begin{cases} \widehat{\beta}(2, j, b) & \text{if } i' = d, \\ u \leftarrow \{0,1\}^{\kappa} & \text{otherwise.} \end{cases}$$

  where d denotes the number of partitions in $\pi'_{\mathsf{digest},i,\mathsf{crs}}$.
- Compute $\rho \leftarrow \mathcal{O}^{\mathsf{Hash}'}_{\mathtt{HASH}^*}(\pi'_{\mathsf{digest},i,\mathsf{crs}}, \widehat{\mu}^{(\widehat{\beta}')})$.
- Outputs a program $\rho'[\rho]$ as follows:
  - Takes as input $k = (k_1, \ldots, k_d)$.
  - Execute ρ on k_j, for every $j \in [d]$ in sequence.
  - Output the final output of ρ.

Fig. 26. Hash Opening Obfuscator

  - Output $\rho'[\rho]$, where ρ' was defined in Figure 26.

Security of the above obfuscation scheme follows directly from that of $\mathcal{O}^{\mathsf{Hash}'}_{\mathtt{HASH}^*}$ as the interaction between the parties and the input sequences follow the same semantics in both classes. We skip those details here for brevity.

□

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

C.1 Proof of Security

Lemma 6 (Restated). *If there exists a CP-ABE scheme, a non-interactive UC secure commitment scheme, a puncturable signature scheme, and an R3PO scheme $\mathcal{O}_{p\text{-}MA\text{-}ABE}$ w.r.t. $(\mathcal{G}^1_{p\text{-}MA\text{-}ABE}, \mathcal{Q}_{p\text{-}MA\text{-}ABE})$[24], then there exists a secure p-MA-ABE scheme.*

Proof: We now prove that the construction in Figure 17 is a secure p-MA-ABE scheme satisfying Definition 19

– **Correctness:** Let

$$(\mathsf{st}, \{\mathsf{req}_i\}_{i\in[\mathsf{N}]}) \leftarrow \text{p-MA-ABE.}\mathsf{KeyRequest}(\mathsf{pk}, \mathsf{gid}, \bar{\mathsf{x}})$$

s.t. for each $i \in [\mathsf{N}]$, $\Theta^{\mathsf{gid}}_i(\bar{\mathsf{x}}) = 1$, $\mathsf{req}_i = (\mathsf{gid}, c_i)$ and $\mathsf{st} = (\mathsf{gid}, \bar{\mathsf{x}}, \{d_i\}_{i\in[\mathsf{N}]})$. For each $t \in [\mathsf{N}]$, let

$$\rho_t = \text{p-MA-ABE.}\mathsf{KeyGen}(t, \mathsf{msk}_t, \Theta^{\mathsf{gid}}_t, \mathsf{req}_t, \{\text{p-MA-ABE.}\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}).$$

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

[24] which is also an R3PO w.r.t. $(\mathcal{G}^2_{\text{p-MA-ABE}}, \mathcal{Q}_{\text{p-MA-ABE}})$.

– **Receiver Privacy:** Let

$$(\mathsf{st}, \{\mathsf{req}_i\}_{i\in[\mathsf{N}]}) \leftarrow \text{p-MA-ABE.KeyRequest}(\mathsf{pk}, \mathsf{gid}, \bar{\mathsf{x}})$$

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

– **Security of Encryption:**
We consider Adv to be a stateful PPT machine. **Hybrid** $\mathcal{H}_{m_0}$: corresponds to the MA-ABE security experiment (Figure 16), where the challenger encrypts message m_0.

Hybrid $\mathcal{H}_{m_0}$:

- Receive H from Adv.
- $\forall i \in H$, $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \text{p-MA-ABE.SetupAuth}(1^\kappa)$. Send $\{\mathsf{mpk}_i\}_{i\in\mathcal{H}}$ to Adv.
- Receive $\{\mathsf{mpk}_j\}_{j\in[\mathsf{N}]\setminus H}$ from Adv
- $\forall j \in \mathsf{poly}(1^\kappa)$, receive request of the form $(t, \mathsf{req}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Adv, where $t \in H$. Send ρ, where

$$\rho \leftarrow \text{p-MA-ABE.KeyGen}\left(t, \mathsf{msk}_t, \Theta_t^{\mathsf{gid}}, \mathsf{req}, \{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}\right)$$

- Receive challenge policy ϕ, messages (m_0, m_1) from Adv. Send $ct \leftarrow \text{p-MA-ABE.Encrypt}\left(\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \phi, m_0\right)$ to Adv
- $\forall j \in \mathsf{poly}(1^\kappa)$, receive request of the form $(t, \mathsf{req}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Adv, where $t \in H$. Send ρ, where

$$\rho \leftarrow \text{p-MA-ABE.KeyGen}\left(t, \mathsf{msk}_t, \mathsf{req}, \Theta_t^{\mathsf{gid}}, \{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}\right)$$

- Receive a bit b' from Adv.

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

Adversary Q_k:

- Sample commitment setups in extractable mode: for $i \in H$, $(\mathsf{st}_i, \mathsf{Com.crs}_i) \leftarrow \mathcal{E}_0^{\mathsf{Com}}(1^\kappa)$. Sample public keys: $\forall i \in H$, $(\mathsf{Sig.vk}_i, \mathsf{Sig.sk}_i) \leftarrow \mathsf{Sig.gen}(1^\kappa)$, $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) \leftarrow \mathsf{ABE.Setup}(1^\kappa)$. Send $\{\mathsf{mpk}_i := (\mathsf{Com.crs}_i, \mathsf{Sig.vk}_i, \mathsf{ABE.mpk}_i)\}_{i\in H}$ to Adv. Receive $\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]/H}$ from Adv.
- Send $\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}$ to G_H^1.
- Queries:
  - on receiving keygen query $j < k$ of the form $(t, \mathsf{req}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Adv, where $t \in H$, $\mathsf{req} = \mathsf{gid}, c)$.
    - extract $\bar{\mathsf{x}} \leftarrow \mathcal{E}_1^{\mathsf{Com}}(\mathsf{st}_t, \mathsf{Com.crs}_t, c)$.
    - Fix $\pi_{\mathsf{pp}}^{(\alpha)} \in \mathcal{P}_{\text{p-MA-ABE}}$ where $\mathsf{pp} = (t, \mathsf{Com.crs}_t, c)$, $\alpha = \Theta_t^{\mathsf{gid}}$, and $\mathsf{start} = \sigma^1_{\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \mathsf{gid}}$.
    - Fix $\mu^{(\beta)} \in \mathcal{M}_{\text{p-MA-ABE}}$ where $\beta = (1, \tau, \mathsf{ABE.msk}_t)$, where $\tau = \mathsf{Sig.sign}(\mathsf{Sig.sk}_t, \bar{\mathsf{x}})$.
    - construct $\hat{\rho} \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi_{pp}^{(\alpha)}, \mu^{(\beta)})$. send $\hat{\rho}$ to Adv.
  - on receiving challenge query of the form (ϕ, m_0, m_1) from Adv. Send $ct \leftarrow \text{p-MA-ABE.Encrypt}\left(\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \phi, m_0\right)$ to Adv
- on receiving keygen query k of the form $(t, \mathsf{req}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Adv, where $t \in H$. Send $(t, \mathsf{req}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ as the target query and $(\mathsf{Sig.sk}_t, \mathsf{ABE.msk}_t)$ to G_H^1.
- halt with output: query k, keys $\{(\mathsf{st}_i, \mathsf{Com.crs}_i, \mathsf{Sig.vk}_i, \mathsf{Sig.sk}_i, \mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i)\}_{i\in H}$, adversary state st_{adv}.

Hybrid $\mathcal{H}_k^1$:

- Receive H from Adv.
- Run $\langle G_H^1 : Q_k \rangle$ and get as output $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G; a_Q)$.
- Parse a_G as $\perp$, parse a_Q as $(t, \mathsf{gid}, c, \{\Theta_i^{\mathsf{gid}}\}_{i \in H})$, $\{(\mathsf{st}_k, \mathsf{Com.crs}_k, \mathsf{Sig.vk}_k, \mathsf{Sig.sk}_k, \mathsf{ABE.mpk}_k, \mathsf{ABE.msk}_k)\}_{k \in H}$, $\mathsf{st}_{\mathsf{Adv}}$ and load the state of Adv.
- Construct $\rho \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi^{(\alpha)}, \mu^{(\beta)})$. Send ρ to Adv.
- Queries:
  - on receiving keygen query $j > k$ of the form $(t, \mathsf{req}, \{\Theta_i^{\mathsf{gid}}\}_{i \in H})$ from Adv, where $t \in H$. Send ρ, where
$$\rho \leftarrow \text{p-MA-ABE.KeyGen}\left(t, \mathsf{msk}_t, \Theta_t^{\mathsf{gid}}, \mathsf{req}, \{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}\right)$$
  - on receiving challenge query of the form (ϕ, m_0, m_1) from Adv. Send $ct \leftarrow \text{p-MA-ABE.Encrypt}\left(\{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}, \phi, m_0\right)$ to Adv
- Receive a bit b' from Adv.

Hybrid $\mathcal{H}_k^2$:

- Receive H from Adv.
- Run $\langle G_H^1 : Q_k \rangle$ and get as output $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G; a_Q)$.
- Parse a_G as $\perp$, parse a_Q as $(t, \mathsf{gid}, c, \{\Theta_i^{\mathsf{gid}}\}_{i \in H})$, $\{(\mathsf{st}_k, \mathsf{Com.crs}_k, \mathsf{Sig.vk}_k, \mathsf{Sig.sk}_k, \mathsf{ABE.mpk}_k, \mathsf{ABE.msk}_k)\}_{k \in H}$, $\mathsf{st}_{\mathsf{Adv}}$ and load the state of Adv.
- extract $\bar{\mathsf{x}} \leftarrow \mathcal{E}_1^{\mathsf{Com}}(\mathsf{st}_t, \mathsf{Com.crs}_t, c)$. Fix $\mu^{(\beta)'} \in \mathsf{M}_{\text{p-MA-ABE}}$ with $\beta = (1, \tau, \mathsf{ABE.mpk}_t)$, where $\tau = \mathsf{Sig.sign}(\mathsf{Sig.sk}_t, \mathsf{gid} || \bar{\mathsf{x}})$.
- Construct $\hat{\rho} \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi^{(\alpha)}, \mu^{(\beta)'})$. Send $\hat{\rho}$ to Adv.
- Queries:
  - on receiving keygen query $j > k$ of the form $(t, \mathsf{req}, \{\Theta_i^{\mathsf{gid}}\}_{i \in H})$ from Adv, where $t \in H$. Send ρ, where
$$\rho \leftarrow \text{p-MA-ABE.KeyGen}\left(t, \mathsf{msk}_t, \Theta_t^{\mathsf{gid}}, \mathsf{req}, \{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}\right)$$
  - on receiving challenge query of the form (ϕ, m_0, m_1) from Adv. Send $ct \leftarrow \text{p-MA-ABE.Encrypt}\left(\{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}, \phi, m_0\right)$ to Adv
- Receive a bit b' from Adv.

Indistinguishability of $\mathcal{H}_{m_0}$ and $\mathcal{H}_\ell^2$: we have $\mathcal{H}_{m_0} \equiv \mathcal{H}_1^1$, and $\forall k \in [\ell]$, hybrids $\mathcal{H}_{k-1}^2 \equiv \mathcal{H}_k^1$, redrawing of boundaries or rephrasing. Now, we prove that $\forall k \in [\ell]$, $\mathcal{H}_k^1 \approx \mathcal{H}_k^2$. Invoking the R3PO security, there exists extractor $\mathcal{E}$ and simulator Sim s.t. in the experiment $\langle G_H^1 : Q_k | \mathcal{E} \rangle$, $\mathcal{E}$ outputs (a_Q, Π, X^*) and $\rho_k \approx \mathsf{Sim}(\pi, \mu, \Pi, \{\mu^{(\beta)}(\sigma)\}_{\sigma \in \text{REACH}_\Pi(X^*)})$. But, $\{\mu^{(\beta)}(\sigma)\}_{\sigma \in \text{REACH}_\Pi(X^*)} = \{\mu^{(\beta)'}(\sigma)\}_{\sigma \in \text{REACH}_\Pi(X^*)}$. Thus, $\rho_k \approx \hat{\rho}_k$.

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

Adversary Q_1':

- Sample commitment setup: $\forall i \in H$, $(\mathsf{st}_i, \mathsf{Com.crs}_i) \leftarrow \mathcal{E}_0^{\mathsf{Com}}(1^\kappa)$, abe keys: $\forall k \in H$, $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) \leftarrow \mathsf{ABE.Setup}(1^\kappa)$. Send $\{(\mathsf{Com.crs}_i, \mathsf{ABE.mpk}_i)\}_{i \in H}$ to G_H^2. Receive $\{\mathsf{Sig.vk}_i\}_{i \in H}$ from G_H^2 and send to Adv. Receive $\{\mathsf{mpk}_i\}_{i \in [\mathsf{N}] \setminus H}$ from Adv and send to G_H^2.

- on receiving keygen query 1 of the form $(t, \mathsf{gid}, c, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Adv, where $t \in H$. Extract $\bar{\mathsf{x}} \leftarrow \mathcal{E}_1^{\mathsf{Com}}(\mathsf{st}_t, c)$, send $(t, \mathsf{gid}, c, \bar{\mathsf{x}}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ as a query to G_H^2, receive $\tau = \{\tau_i\}_{i\in H}$ from G_H^2 and send $(t, \mathsf{gid}, c, \bar{\mathsf{x}}, \tau_t, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ as the target request to G_H^2.
- halt with output $\{(\mathsf{st}_i, \mathsf{Com.crs}_i, \mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, \{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]\setminus H}, st_{adv}$

- **Adversary** Q'_k:
- Let $\mathcal{E}_{k-1}$ be a reach extractor for Q'_{k-1} w.r.t. generator G_H^2.
- Internally run $Q'_{k-1}|\mathcal{E}_{k-1}$.
- Receive $\{(\mathsf{Com.crs}_i, \mathsf{ABE.mpk}_i)\}_{i\in H}$ from Q'_{k-1} and forward it to G_H^2. Receive $\{\mathsf{Sig.vk}_i\}_{i\in H}$ from G_H^2 and forward it to Q'_{k-1}. Receive $\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]\setminus H}$ from Q'_{k-1} and forward it to G_H^2.
- Queries:
  - on receiving query $j < k$ of the form $(t, \mathsf{gid}, c, \bar{\mathsf{x}}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Q'_{k-1}, forward it to G_H^2, receive τ from G_H^2 and forward it to Q'_{k-1}
  - on receiving target query $j = k-1$ of the form $(t, \mathsf{gid}, c, \bar{\mathsf{x}}, \tau_t, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Q'_{k-1}:
    - Get (a_Q, Π, X^*) as the output of $Q'_{k-1}|\mathcal{E}_{k-1}$ after it halts.
    - Parse a_Q as $\{(\mathsf{st}_i, \mathsf{Com.crs}_i, \mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, \{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]\setminus H}, st_{adv}\Big)$ and load the state of Adv.
    - Construct $\widetilde{\rho} \leftarrow \mathsf{Sim}_{\text{p-MA-ABE}}(\pi, \mu, \Pi, \{\mu^{(\beta)}(\sigma)\}_{\sigma\in \text{REACH}_\Pi(X^*)})$. Send $\widetilde{\rho}$ to Adv.
  - on receiving challenge query of the form (ϕ, m_0, m_1) from Adv. Send $ct \leftarrow \text{p-MA-ABE.}\mathsf{Encrypt}\left(\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \phi, m_0\right)$ to Adv
  - on receiving query $j = k$ of the form $(t, \mathsf{gid}, c, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Adv.
    - Extract $\bar{\mathsf{x}} \leftarrow \mathcal{E}_1^{\mathsf{Com}}(\mathsf{st}_t, c)$ and send $(t, \mathsf{gid}, c, \bar{\mathsf{x}}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ as a request to G_H^2. Receive $\tau = \{\tau_i\}_{i\in H}$ from G_H^2. Send $(t, \mathsf{gid}, c, \bar{\mathsf{x}}, \tau_t, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ as the target request to G^2p-MA-ABE.
    - halt with output $\{(\mathsf{st}_i, \mathsf{Com.crs}_i, \mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, \{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]\setminus H}, st_{adv}$.

Hybrid $\mathcal{H}_k^3$:

- Receive H from Adv.
- Run $\langle G_H^2 : Q'_k\rangle$ and get as output $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G; a_Q)$.
- Parse a_G as $\{(\mathsf{Sig.vk}_i, \mathsf{Sig.sk}_i)\}_{i\in H}$, parse a_Q as $\{(\mathsf{st}_i, \mathsf{Com.crs}_i, \mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, \{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]\setminus H}, st_{adv}$ and load the state of Adv.
- Construct $\rho \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi^{(\alpha)}, \mu^{(\beta)})$. Send ρ to Adv.
- Queries:
  - on receiving keygen query $j > k$ of the form $(t, \mathsf{req}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$ from Adv, where $t_j \in H$ and $\mathsf{req} = (\mathsf{gid}, c)$:
    - extract $\bar{\mathsf{x}} \leftarrow \mathcal{E}_1^{\mathsf{Com}}(\mathsf{st}_t, \mathsf{Com.crs}_t, c)$.
    - Fix $\pi_{\mathsf{pp}}^{(\alpha)} \in \mathcal{P}_{\text{p-MA-ABE}}$ where $\mathsf{pp} = (t, \mathsf{Com.crs}_t, c), \alpha = \Theta_t^{\mathsf{gid}}$, and $\mathsf{start} = \sigma^1_{\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \mathsf{gid}}$.
    - Fix $\mu^{(\beta)} \in \mathcal{M}_{\text{p-MA-ABE}}$ where $\beta = (1, \tau_t, \mathsf{ABE.msk}_t)$, where $\tau = \mathsf{Sig.sign}(\mathsf{Sig.sk}_t, \bar{\mathsf{x}})$.
    - construct $\hat{\rho} \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi_{pp}^{(\alpha)}, \mu^{(\beta)})$. send $\hat{\rho}$ to Adv.
  - on receiving challenge query of the form (ϕ, m_0, m_1) from Adv. Send $ct \leftarrow \text{p-MA-ABE.}\mathsf{Encrypt}\left(\{\mathsf{mpk}_i\}_{i\in[\mathsf{N}]}, \phi, m_0\right)$ to Adv
- Receive a bit b' from Adv.

Hybrid $\mathcal{H}^4_k$:

- Receive H from Adv.
- Run $\langle G^2_H : Q'_k | \mathcal{E}_k \rangle$ and get as output $(\pi^{(\alpha)}, \mu^{(\beta)}, a_G; a_Q, \Pi, X^*)$.
- Parse a_G as $\{(\mathsf{Sig.vk}_i, \mathsf{Sig.sk}_i)\}_{i \in H}$, parse a_Q as $\{(\mathsf{st}_i, \mathsf{Com.crs}_i, \mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, \{\mathsf{mpk}_i\}_{i \in [\mathsf{N}] \setminus H}, st_{adv}$ and load the state of Adv.
- Construct $\widetilde{\rho} \leftarrow \mathsf{Sim}_{\text{p-MA-ABE}}(\pi, \mu, \Pi, \{\mu^{(\beta)}(\sigma)\}_{\sigma \in \text{REACH}_\Pi(X^*)})$. Send $\widetilde{\rho}$ to Adv.
- Queries:
  - on receiving keygen query $j > k$ of the form $(t, \mathsf{req}, \{\Theta^{\mathsf{gid}}_i\}_{i \in H})$ from Adv, where $t \in H$ and $\mathsf{req} = (\mathsf{gid}, c)$:
    - extract $\bar{\mathsf{x}} \leftarrow \mathcal{E}^{\mathsf{Com}}_1(\mathsf{st}_t, \mathsf{Com.crs}_t, c)$.
    - Fix $\pi^{(\alpha)}_{\mathsf{pp}} \in \mathcal{P}_{\text{p-MA-ABE}}$ where $\mathsf{pp} = (t, \mathsf{Com.crs}_t, c), \alpha = \Theta^{\mathsf{gid}}_t$, and $\mathsf{start} = \sigma^1_{\{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}, \mathsf{gid}}$.
    - Fix $\mu^{(\beta)} \in \mathcal{M}_{\text{p-MA-ABE}}$ where $\beta = (1, \tau_t, \mathsf{ABE.msk}_t)$, where $\tau_t = \mathsf{Sig.sign}(\mathsf{Sig.sk}_t, \bar{\mathsf{x}})$.
    - construct $\hat{\rho} \leftarrow \mathcal{O}_{\text{p-MA-ABE}}(\pi^{(\alpha)}_{pp}, \mu^{(\beta)})$. send $\hat{\rho}$ to Adv.
  - on receiving challenge query of the form (ϕ, m_0, m_1) from Adv. Send $ct \leftarrow$ p-MA-ABE.$\mathsf{Encrypt}\left(\{\mathsf{mpk}_i\}_{i \in [\mathsf{N}]}, \phi, m_0\right)$ to Adv
- Receive a bit b' from Adv.

Indistinguishability of $\mathcal{H}_{m_0}$ and $\mathcal{H}^4_\ell$: we have $\mathcal{H}^2_\ell \equiv \mathcal{H}^3_1$ and $\forall k \in [\ell]$, hybrids $\mathcal{H}^4_{k-1} \equiv \mathcal{H}^3_k$, redrawing of boundaries or rephrasing. Now, we prove that that $\forall k \in [\ell]$, $\mathcal{H}^3_k \approx \mathcal{H}^4_k$. Invoking the R3PO security, in the experiment $\langle G^2_H : Q_k | \mathcal{E}_k \rangle$, $\mathcal{E}_k$ outputs (a_Q, Π, X^*) s.t. $\hat{\rho}_k \approx \mathsf{Sim}(\pi, \mu, \Pi, \{\mu^{(\beta)}(\sigma)\}_{\sigma \in \text{REACH}_\Pi(X^*)}$.
Note that, in the experiment $\langle G^2_H : Q'_k | \mathcal{E}_k \rangle$, if the target request is of the form $(t, \mathsf{gid}, c_t, \bar{\mathsf{x}}, \tau_t)$ and the extractor $\mathcal{E}_k$ outputs a_Q, Π, X^*, then we have:

- if $\forall i \in H, \Theta^{\mathsf{gid}}_i(\bar{\mathsf{x}}) = 1$, then $\sigma^{3,\mathsf{N}}_{\mathsf{pk},\mathsf{gid},\bar{\mathsf{x}}} \in \text{REACH}_\Pi(X^*)$. Then, $\{\mu^{(\beta)}(\sigma)\}_{\sigma \in \text{REACH}_\Pi(X^*)} = \{\tau_t, \mathsf{ABE.KeyGen}(\mathsf{ABE.msk}_t, \bar{\mathsf{x}})\}$.
- otherwise, $\{\mu^{(\beta)}(\sigma)\}_{\sigma \in \text{REACH}_\Pi(X^*)} = \{\tau_t\}$.

Hybrid $\mathcal{H}_{m_1}$: We now replace the challenge message m_0 in the ciphertext ct with the challenge message m_1 as follows. Let the additive secret sharing of m_0 be $\{s_1, \ldots, s_{\mathsf{N}}\}$; the ABE ciphertexts be $\{\mathsf{ABE.ct}_0, \ldots, \mathsf{ABE.ct}_{\mathsf{N}}\}$, where $\forall i \in [\mathsf{N}]$, $\mathsf{ABE.ct}_i \leftarrow \mathsf{ABE.Encrypt}(\mathsf{ABE.mpk}_i, \phi, s_i)$; and the ciphertext p-MA-ABE.ct $= \{\mathsf{ABE.ct}_0, \ldots, \mathsf{ABE.ct}_{\mathsf{N}}\}$. Let $H_0 \in H$ be an honest authority. We replace the secret share s_{H_0} with $s_{H_0} + (m_1 - m_0)$ and compute ciphertext $\tilde{\mathsf{ct}}_{H_0}$. Indistinguishability of $\mathcal{H}_{m_1}$ and $\mathcal{H}^2_{N'}$ follows from the indistinguishability security of the underlying ABE encryption scheme at authority A_{H_0}.
Overall: we get that the hybrids $\mathcal{H}_{m_0}$ and $\mathcal{H}_{m_1}$ are indistinguishable. But, these hybrids correspond to the MA-ABE security experiment in Figure 16. Thus, the above scheme satisfies MA-ABE security.

□

C.2 Modeling the Key-Component as an R3PO

We define two generator classes that will be useful to prove security of the scheme in Figure 17. 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 $\mathbb{A}_i$ in each reactive-program obfuscation for a key-request $\mathsf{gid} \| c$ with a signature τ on $\bar{\mathsf{x}}$ s.t. the commitment c can only be opened to $\bar{\mathsf{x}}$ (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 $\mathbb{A}_i$ in the obfuscation with: either the ABE key for $\bar{\mathsf{x}}$ (if the attribute-granting policies of all the honest authorities agree) or empty key.

Program class : Each program $(\pi^{(\alpha)}_{\mathsf{pp}}, \mu^{(\beta)})$ in $\mathcal{P}_{\text{p-MA-ABE}}$ is specified as follows:

– public parameter $\mathsf{pp} = (t, \mathsf{Com.crs}, c)$ (hardcoded in the program)
– start state $\mathsf{start} = \sigma^1_{\mathsf{pk},\mathsf{gid}}$, where $\mathsf{pk} = \{\mathsf{Sig.vk}_k\}_{k \in [n]}$

– transition function $\pi_{\mathsf{pp}}^{(\alpha)}$, where $\alpha = \Theta^{\mathsf{gid}}$, is defined as:

$$\pi_{\mathsf{pp}}^{(\alpha)}\left(\sigma_{\mathsf{st}}^{t}, y\right) = \begin{cases} \sigma^{2}_{\mathsf{pk},\mathsf{gid},\mathsf{gid}',\bar{\mathsf{x}}} & \text{where } t = 1, \mathsf{st} = (\mathsf{pk}, \mathsf{gid}), \\ & \text{if } \mathsf{Com.Open}(\mathsf{Com.crs}, c, y) = \mathsf{gid}'\|\bar{\mathsf{x}}. \\ \sigma^{3,0}_{\mathsf{pk},\mathsf{gid},\bar{\mathsf{x}}} & \text{where } t = 2, \mathsf{st} = (\mathsf{pk}, \mathsf{gid}, \mathsf{gid}', \bar{\mathsf{x}}), \\ & \text{if } \Theta^{\mathsf{gid}}(\bar{\mathsf{x}}) = 1, \mathsf{gid}' = \mathsf{gid}. \\ \sigma^{3,k}_{\mathsf{pk},\mathsf{gid},\bar{\mathsf{x}}} & \text{where } t = (3, k-1), k \in [\mathsf{N}], \mathsf{st} = (\mathsf{pk}, \mathsf{gid}, \bar{\mathsf{x}}), y = (\tau_k, m_k), \\ & \text{if } \mathsf{Sig.verify}(\mathsf{Sig.vk}_k, \tau_k, m_k) = 1, m_k = \mathsf{gid}\|\bar{\mathsf{x}}. \end{cases}$$

– message function $\mu^{(\beta)} \in \mathcal{M}_{\text{p-MA-ABE}}$, with $\beta = (b, \mathsf{ss}, \mathsf{ABE.msk})$, for $b \in \{0,1\}$ and ss of the form:

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

is defined as:

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

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

Generator Class $\mathcal{G}^1_{\text{p-MA-ABE}}$: 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 $\{\mathsf{mpk}\}_{i\in[\mathsf{N}]}$ of the authorities from Q.
- it finally receives a target request of the form $(t, \mathsf{gid}, c, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$, where $t \in H$ and $(\mathsf{Sig.sk}_t, \mathsf{ABE.msk}_t)$.
- it halts with output $(\pi_{\mathsf{pp}}^{(\alpha)}, \mu^{(\beta)}) \in \mathcal{P}_{\text{p-MA-ABE}}$, where $\mathsf{pp} = (t, \mathsf{Com.crs}_t, c)$, $\alpha = \Theta_t^{\mathsf{gid}}$, $\mathsf{start} = \sigma^1_{\mathsf{pk},\mathsf{gid}}$ and $\beta = (0, \mathsf{Sig.sk}_t, \mathsf{ABE.msk}_t)$.

Generator Class $\mathcal{G}^2_{\text{p-MA-ABE}}$: 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 $\{\mathsf{Com.crs}_i\}_{i\in H}$, ABE key pairs $\{(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i)\}_{i\in H}$ from Q.
- it samples signature key pairs $\{(\mathsf{Sig.vk}_i, \mathsf{Sig.sk}_i)\}_{i\in H}$ and sends $\{\mathsf{Sig.vk}_i\}_{i\in H}$ to Q.
- it receives signature verification keys $\{\mathsf{Sig.vk}_i\}_{i\in[\mathsf{N}]\setminus H}$ of the corupt authorities from Q.
- For $k \in \mathsf{poly}(1^\kappa)$, it receives query of the form $(t, \mathsf{gid}, \bar{\mathsf{x}}, \{\Theta_i^{\mathsf{gid}}\}_{i\in H})$, and sends $\{\tau_i\}_{i\in H}$, where $\forall i \in H$:

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

it also updates[25] the signature key for authorities $i \in H$ as:

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

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

[25] 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.

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

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 (Appendix B.1) and signature-checking (Appendix B.2) R3PO in the library. Thus, we have:

We prove that the MA-ABE generator classes defined above decompose to a library of one-step program program generators.

Lemma 15. *$(\mathcal{G}^1_{p\text{-}MA\text{-}ABE}, \mathcal{Q})$ decomposes into*

$$\boldsymbol{\mathcal{L}} = (\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_{3,0}, \ldots, \mathcal{L}_{3,\mathsf{N}}),$$

where:

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

Proof: We prove that $\forall G^1_H \in \mathcal{G}^1_{\text{p-MA-ABE}}$, G^1_H decomposes to $\boldsymbol{\mathcal{L}}$.
Case $i = 1$:
At this partition, we prove that $(G^1_H, \mathcal{Q})$ decomposes to $(\mathcal{G}^{\mathsf{Com}}_{\text{COMMIT}}, \mathcal{Q})$ (with the appropriate prefix and partition function, that we skip here; refer Section 6.5):

- **Defining** J: It extracts pk, gid, pp from the transcript of the interaction and halts with output $(\widehat{\pi}^{(\alpha)}_{\sigma^1_{st}}, \widehat{\mu}^{(\widehat{\beta})})$.
- **Defining** Z: It interacts with Q by internally running G^1_H. Finally, it outputs G^1_H'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 $(G^1_H, \mathcal{Q})$ decomposes to $(\mathcal{G}^{\Sigma}_{\epsilon}, \mathcal{Q})$ (with the appropriate prefix and partition function, that we skip here; refer Section 6.5):

- **Defining** J: It receives σ_1, σ_2 from W, samples a message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{p-MA-ABE},1}}$ and halts with output $(\widehat{\pi}^{(\alpha)}_{\sigma_2}[\sigma_1], \widehat{\mu}^{(\widehat{\beta})})$.
- **Defining** Z: It interacts with Q by internally running G^1_H. Finally, it outputs G^1_H's output.
- **Defining** W: It extracts σ_1, σ_2 from the transcript of the interaction and the output of a 1-partial reach extractor $\mathcal{E}$ (which extracts gid' from c). It sends σ_1, σ_2 on the bottom channel.

Case $i = (3,a)$, if $a+1 \in [\mathsf{N}]$:
At this partition, we prove that $(G^1_H, \mathcal{Q})$ decomposes to $(\mathcal{G}^{\mathsf{Sig}}_{\text{QSIGN}}, \mathcal{Q})$ (with the appropriate prefix and partition function, that we skip here; refer Section 6.5):

- **Defining** J: It extracts $\sigma = \sigma^{3,i}_{\mathsf{pk},\mathsf{gid},\bar{\mathsf{x}}}$ from the transcript of the interaction, samples a message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{p-MA-ABE},3+i}}$ and halts with output $(\widehat{\pi}^{(\alpha)}_{\sigma}, \widehat{\mu}^{(\widehat{\beta})})$.
- **Defining** Z: It interacts with Q by internally running G^1_H. Finally, it outputs G^1_H's output.
- **Defining** W: It extracts $\mathsf{Sig.vk}_{i+1}$, gid, $\bar{\mathsf{x}}$ from the transcript of the interaction and sends $\mathsf{Sig.vk}_{i+1}$, $\mathsf{gid}\|\bar{\mathsf{x}}$ on the bottom channel.

□

Lemma 16. *$(\mathcal{G}^2_{p\text{-}MA\text{-}ABE}, \mathcal{Q}_{p\text{-}MA\text{-}ABE})$ decomposes into $\overline{\boldsymbol{\mathcal{L}}} = \{\boldsymbol{\mathcal{L}}_H\}_H$, where, for each H (set of honest authorities),*

$$\boldsymbol{\mathcal{L}}_H = (\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_{3,0}, \ldots, \mathcal{L}_{3,\mathsf{N}}),$$

and $\forall i \in [3+\mathsf{N}]$*:*

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

Proof: We prove that $\forall G^2_H \in \mathcal{G}^2_{\text{p-MA-ABE}}$, G^2_H decomposes to $\boldsymbol{\mathcal{L}}_H$. The first three cases are similar to Lemma 15.

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

At this partition, we prove that $(G^2_H, \mathcal{Q})$ decomposes to $(\mathcal{G}^{\mathsf{Sig}}_{\text{SIGN}}, \mathcal{Q})$ (with the appropriate prefix and partition function, that we skip here; refer Section 6.5):

- **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 τ_j. Finally Q sends the target message m on the top channel. It extracts $\mathsf{gid}, \bar{\mathsf{x}}$ 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 $\mathsf{gid}, \bar{\mathsf{x}}$ from W as the target message, samples a message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\text{p-MA-ABE}}, 2+a}$ and outputs $(\widehat{\pi}^{(\alpha)}_{\sigma^{3,a}_{st}}, \widehat{\mu}^{(\widehat{\beta})})$.
- **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).

□

We now prove the main lemma.

Lemma 17. *If there exist R3PO schemes w.r.t.* $(\mathcal{G}^{\mathsf{Com}}_{COMMIT}, \mathcal{Q}^{\mathsf{Com}}_{COMMIT}, \mathring{\mathcal{P}}_{COMMIT})$ *and* $(\mathcal{G}^{\mathsf{Sig}}_{SIGN}, \mathcal{Q}^{\mathsf{Sig}}_{SIGN}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{SIGN})$*, where* Com *is a non-interactive UC-secure commitment scheme and* Sig *is a puncturable signature scheme, then there exists a compiler* $\mathcal{O}_{p\text{-}MA\text{-}ABE}$ *which is an R3PO scheme w.r.t.* $(\mathcal{G}^1_{p\text{-}MA\text{-}ABE}, \mathcal{Q}_{p\text{-}MA\text{-}ABE})$*, and also an R3PO w.r.t.* $(\mathcal{G}^2_{p\text{-}MA\text{-}ABE}, \mathcal{Q}_{p\text{-}MA\text{-}ABE})$.

Proof: If there exists a semi-honest secure OTSE scheme, then there exists a PPT program $\mathcal{O}_{\mathsf{Sig}}$ s.t. $\mathcal{O}_{\mathsf{Sig}}$ is a R3PO scheme w.r.t. $(\mathcal{G}^{\mathsf{Sig}}_{\text{SIGN}}, \mathcal{Q}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{\text{SIGN}})$ (Lemma 2) and $\mathcal{O}_{\mathsf{Sig}}$ is a R3PO scheme w.r.t. $(\mathcal{G}^{\mathsf{Sig}}_{\text{QSIGN}}, \mathcal{Q}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{\text{QSIGN}})$ (Lemma 3).

Now, from our compiler (Theorem 1), Lemma 15 and Lemma 16 and Lemma 3, 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. $(\mathcal{G}^1_{\text{p-MA-ABE}}, \mathcal{Q}_{\text{p-MA-ABE}})$ and $(\mathcal{G}^2_{\text{p-MA-ABE}}, \mathcal{Q}_{\text{p-MA-ABE}})$. □

D 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 6.2). In fact, our construction readily extends to identity-based *functional* encryption, which may be of further interest.

D.1 Definition for IBFE

Definition 20 (Identity Based Functional Encryption (IBFE)). An IBFE scheme for a functionality $\mathcal{F}$ defined over (K, M) consists of the following PPT algorithms:

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

- $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id},f}, c) \to y$: On input a function key $\mathsf{sk}_{\mathsf{id},f}$ and ciphertext c, outputs a message y.

The following properties are required from the above algorithms.

– **Completeness**: For all security parameters κ, $\forall \mathsf{id} \in \{0,1\}^n, \forall f \in \mathsf{K}, \forall m \in \mathsf{M}$ it holds that, if $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$, $\mathsf{sk}_{\mathsf{id},f} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, f)$, then:

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id},f}, \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}, m)) = \mathcal{F}(f, m)\right] = 1$$

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

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

where $\mathrm{IND}^{\mathrm{IBFE}}$ is shown in Figure 27.

◁

Experiment $\mathbf{IND}^{\mathbf{IBFE}}$

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

– $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$.
– $(s_{\mathcal{A}_1}, \mathsf{id}^*, m_0, m_1) \leftarrow \mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk})$.
– $b \leftarrow \{0,1\}$.
– $c \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}^*, m_b)$.
– $b^* \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(s_{\mathcal{A}_1}, \mathsf{mpk}, c)$.
– If $\mathcal{A}_1$ or $\mathcal{A}_2$ queried key for an (id^*, f^*) such that $\mathcal{F}(f^*, m_0) \neq \mathcal{F}(f^*, m_1)$ or queried id^* for more than one key, output a random bit $b \leftarrow \{0,1\}$.
Else, output 1 if $b' = b^*$ else 0.

Fig. 27. IBFE Security Experiment.

D.2 Construction for IBFE

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

- a puncturable signature scheme (Definition 11) Sig, with message space $\mathsf{M} = \{0,1\}^{m+o+1}$, where m is the length of identity strings and o is the length of function keys in IBFE.
- an R3PO scheme $\mathcal{O}^{\mathsf{Sig}}_{\mathrm{SIGN}}$ w.r.t. $(\mathcal{G}^{\mathsf{Sig}}_{\mathrm{SIGN}}, \mathcal{Q}^{\mathsf{Sig}}_{\mathrm{SIGN}}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{\mathrm{SIGN}})$ (defined in Section 6.2).

Lemma 18. *If there exists a puncturable signature scheme* Sig *and an R3PO scheme w.r.t.* $(\mathcal{G}^{\mathsf{Sig}}_{SIGN}, \mathcal{Q}^{\mathsf{Sig}}_{SIGN}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{SIGN})$*, then there exists a secure IBFE scheme.*

Proof: We prove that the scheme in Figure 28 is a secure IBFE scheme satisfying Definition 20. It uses an R3PO scheme $\mathcal{O}^{\mathsf{Sig}}_{\mathrm{SIGN}}$ w.r.t. $(\mathcal{G}^{\mathsf{Sig}}_{\mathrm{SIGN}}, \mathcal{Q}^{\mathsf{Sig}}_{\mathrm{SIGN}}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{\mathrm{SIGN}})$ as described in Section 6.2.1 (note that a generator in $\mathcal{G}^{\mathsf{Sig}}_{\mathrm{SIGN}}$ is denoted as $\mathsf{H}^{\mathsf{Sig}}_{\mathrm{SIGN}}[Z]$ there).

Parameter: Let κ be the security parameter.
Let $\mathsf{Sig} = (\mathsf{Sig.gen}, \mathsf{Sig.sign}, \mathsf{Sig.verify}, \mathsf{Sig.punct}, \mathsf{Sig.psign})$ be a puncturable signature scheme.
Let $\mathcal{O}^{\mathsf{Sig}}_{\text{SIGN}}$ be an R3PO scheme w.r.t. $(\mathcal{G}^{\mathsf{Sig}}_{\text{SIGN}}, \mathcal{Q}^{\mathsf{Sig}}_{\text{SIGN}}, \mathring{\mathcal{P}}^{\mathsf{Sig}}_{\text{SIGN}})$.

IBFE scheme:

- $\mathsf{Setup}(1^\kappa)$:
  $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.gen}(1^\kappa)$
  output $(\mathsf{mpk}, \mathsf{msk}) := (\mathsf{Sig.vk}, \mathsf{Sig.sk})$.

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

- $\mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}, m)$:
  parse mpk as $\mathsf{Sig.vk}$
  let $\mu_{\mathsf{id},\mathcal{F}}$ be as follows:

$$\mu^{(m)}_{\mathsf{id},\mathcal{F}}(\sigma^i_x) = \begin{cases} \mathcal{F}(f,m) & \text{if } i = 2 \text{ and } x = \mathsf{id}\|f, \\ \bot & \text{otherwise.} \end{cases}$$

  output $\mathcal{O}^{\mathsf{Sig}}_{\text{SIGN}}\big(\pi^{(\cdot)}_{\mathsf{id}}[\sigma^1_{\mathsf{Sig.vk}}], \mu^{(m)}_{\mathsf{id},\mathcal{F}}, \mathcal{I}_{\text{SIGN}}\big)$

- $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id},f}, \mathsf{ct})$:
  parse $\mathsf{sk}_{\mathsf{id},f}$ as (τ, x)
  parse ct as ρ
  output $m \leftarrow \rho((\tau, x))$

Fig. 28. IBFE from R3PO for Signatures

Let $\mathsf{Adv} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary for the IBFE security experiment in Figure 27 with advantage α. We give a reduction to the R3PO security (Definition 16) of $\mathcal{O}^{\mathsf{Sig}}_{\text{SIGN}}$ by constructing a generator $G \in \mathcal{G}^{\mathsf{Sig}}_{\text{SIGN}}$, an adversary $Q \in \mathcal{Q}^{\mathsf{Sig}}_{\text{SIGN}}$ and a PPT distinguisher D'. Recall that, G is of the form $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}\|Z$ for a fixed $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}$ (in the library) and PPT Z. We now describe Z, Q, D' and the interaction:

- Q internally runs $\mathcal{A}_1$ in a straightline black-box way.
- Q receives $\mathsf{Sig.vk}$ from $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}$ and forwards it to $\mathcal{A}_1$.
- KeyGen queries: Q receives polynomial many key queries from $\mathcal{A}_1$ of the form (id_i, f_i), for each such query, Q sends query for signature of $\mathsf{id}_i\|f_i$ to $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}$ and forwards the signature response to $\mathcal{A}_1$.
- Challenge: Q receives the challenge $(\mathsf{id}^*, m_0, m_1)$ from $\mathcal{A}_1$ and forwards it to Z. Z sends id^* as the target mpre to $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}$.
- Q receives the punctured key sk (punctured on all signature queries above and on id^*) from $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}$.
- Z samples $b \leftarrow \{0,1\}$, sets $m := m_b$ and sends $\mu^{(m)}_{\mathsf{id}^*,\mathcal{F}}$ to $\mathsf{H}^{\mathsf{Sig}}_{\text{SIGN}}$ as the message function.
- Z outputs $a_G = b$ as its aux output, Q outputs $a_Q = (\mathsf{id}^*, \mathsf{sk}, m_0, m_1)$ as its aux output.

At the end of the interaction, D' takes the output of the execution (ρ, a_G, a_Q) as its input; where ρ is the real obfuscation for the execution $\text{REAL}(G, Q)$ and the simulated obfuscation output by Sim for the ideal execution $\text{IDEAL}(G, Q\hat{|}\mathcal{E})$. D' internally runs $\mathcal{A}_2$ in a straightline black-box way and behaves as follows.

- D' parses a_G as b, parses a_Q as $(\mathsf{id}^*, \mathsf{sk}, m_0, m_1)$ and sends ρ to $\mathcal{A}_2$ as the challenge ciphertext.
- KeyGen queries: D' responds to polynomial many keygen queries from $\mathcal{A}_2$ using the punctured key sk.
- D' finally receives a bit b' from $\mathcal{A}_2$. Let the single keygen query on the target id by $\mathcal{A}_1 \cup \mathcal{A}_2$ be (id^*, f). If there are multiple queries or if $\mathcal{F}(k, m_0) \neq \mathcal{F}(k, m_1)$, 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 $\mathcal{A}_2$ get a simulated obfuscation $\tilde{\rho}$ that is independent of b, m_0 and m_1 (except for the output $\mathcal{F}(k, m_0) = \mathcal{F}(k, m_1)$). Thus, it outputs 1 with probability 1/2. Now, if α is non-negligible, then D' breaks R3PO security. □

E MPC from R3PO for Commitment-Opening

In this section, we give our 2-round MPC construction, which is a simpler rederivation of the results in [7, 30] using an R3PO scheme for commitment opening (see Appendix B.1) and our composition theorem.

E.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 ϕ, 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 Φ 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 $\{\mathsf{G}_w\}_{w \in A_i}$ and a string z_i such that $|z_i| = |x_i|$. It then broadcasts z_i. A common state vector s is initialized as $z_1 \| \ldots \| z_n$ (at this point s will be L bits long).
  - The second phase consists of T rounds. In round $w - L$, if $\phi_w = (i, u, v)$, party P_i broadcasts the bit $\mathsf{s}_w = \mathsf{G}_w(\mathsf{s}_u, \mathsf{s}_v)$, 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 $\mathsf{G}_w(\mathsf{s}_u, \mathsf{s}_v)$ and outputs it.

Just like is done in [30], 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 $\{\mathsf{G}_w\}_{w \in A_i}$ maybe arbitrary for corrupt P_i) and passive corruption in Phase 2.

E.2 MPC Construction

We will build a 2-round MPC protocol assuming a multi-round conforming protocol Φ for the same functionality, a commitment scheme, and an R3PO scheme $\mathcal{O}^{\mathsf{Com}}_{\text{COMMIT}}$ w.r.t. $(\mathcal{G}^{\mathsf{Com}}_{\text{COMMIT}}, \mathcal{Q}^{\mathsf{Com}}_{\text{COMMIT}}, \mathring{\mathcal{P}}_{\text{COMMIT}})$.[26] We only need a commitment scheme where the messages to be committed are bits i.e., $\mathsf{M} = \{0,1\}$. First, we must 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 $\mathcal{P}^{\mathsf{Com},i}_{\mathsf{MPC}}$: Each program in the program family is parameterized by a conforming protocol Φ, a vector of common random strings $\mathsf{crs} = \{\mathsf{crs}_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$, a vector of commitments $\mathsf{c} = \{c_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$ and an index i. The state space is defined as

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

[26] Defined in Appendix B.1.

A program $\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}[\sigma^0_{\mathsf{s}^0}] \in \mathcal{P}^{\mathsf{Com},i}_{\mathsf{MPC}}$, where $\alpha = \{\mathsf{G}_w\}_{w\in A_i}$, is defined by the transition function:

$$\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}[\sigma^0_{\mathsf{s}^0}](\sigma^{w-L-1}_{\mathsf{s}}, x) = \begin{cases} \sigma^{w-L}_{\mathsf{s}'} & \text{if } \phi_w = (i,u,v), \\ & \text{where } \mathsf{s}' = \mathsf{s}\|\mathsf{G}_w(\mathsf{s}_u,\mathsf{s}_v). \\ \sigma^{w-L}_{\mathsf{s}'} & \text{if } \phi_w = (i^* \neq i,u,v), \\ & \text{where } \mathsf{s}' = \mathsf{s}\|\mathsf{Com.Open}(\mathsf{crs}_{w,\mathsf{s}_u,\mathsf{s}_v},(c_{w,\mathsf{s}_u,\mathsf{s}_v},x)). \\ \sigma^{w-L-1}_{\mathsf{s}} & \text{otherwise.} \end{cases}$$

for $w \in \{L:T\}$. We shall also associate a partition function $\mathcal{I}_{\mathsf{MPC}} : \Sigma \to \{0,\ldots,T+1\}$, such that $\mathcal{I}_{\mathsf{MPC}}(\sigma^{w-L}_{\mathsf{s}}) = w-L$, for $w \in \{L,\ldots,L+T+1\}$.

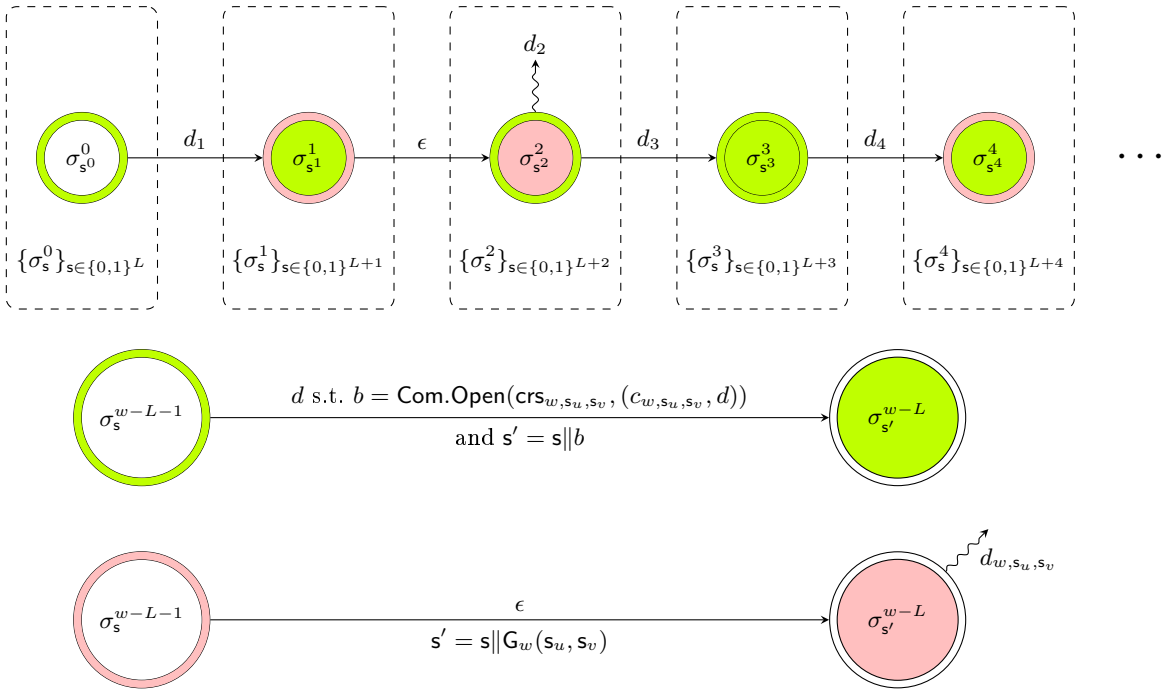

Fig. 29. Top figure shows the transition program $\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}[\sigma^0_{\mathsf{s}^0}]$ produced by P_i, when $L+2, L+5 \in A_i$ and $L+1, L+3, L+4 \notin A_i$. The details of the transitions for w such that $\phi_w = (j,u,v)$, where $j \neq i$ (middle) and $j = i$ (bottom) are also shown.

– Program family $\mathring{\mathcal{P}}_{\mathsf{MPC}}$: This program family is essentially the same as $\mathcal{P}^{\mathsf{Com},i}_{\mathsf{MPC}}$ where Com is a trivial, non-hiding commitment scheme (i.e., $\mathsf{Com.Commit}(m) = m$). In other words, a program $\Pi_{\Phi,\mathsf{x}}[\sigma^0_{\mathsf{s}^0}] \in \mathring{\mathcal{P}}_{\mathsf{MPC}}$ is defined by the transition function:

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

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

– A $(\mathcal{P}^{\mathsf{Com},i}_{\mathsf{MPC}}, \widehat{\mathcal{M}}_{\mathcal{I}_{\mathsf{MPC}},1})$-Generator Class $\mathcal{G}^{\mathsf{Com},i}_{\mathsf{MPC}}$: This class contains all generators of the form $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}\|Z$, for all PPT Z, where $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$ behaves as follows: It receives $\mathsf{crs} = \{\mathsf{crs}_{w,\gamma,\delta}\}_{w\in\{L:T\},\gamma,\delta\in\{0,1\}}$ from Q. It then

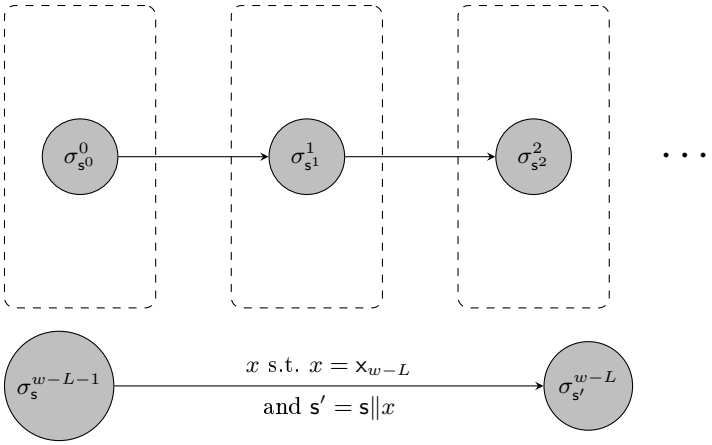

Fig. 30. The top figure denotes the relaxed transition program $\Pi_{\Phi,\mathsf{x}}[\sigma^0_{\mathsf{s}^0}]$, with a particular transition for w s.t. $\phi_w = (\cdot, u, v)$ being shown in the bottom figure.

generates $(z_i, \{\mathsf{G}_w\}_{w\in A_i}) \leftarrow \mathsf{pre}(1^\kappa, i, x_i)$, with x_i being an input to $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$. For every $w \in A_i$, and $\gamma, \delta \in \{0,1\}$, $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$ generates

$$(c_{w,\gamma,\delta}, d_{w,\gamma,\delta}) \leftarrow \mathsf{Com.Commit}(\mathsf{crs}_{w,\gamma,\delta}, \mathsf{G}_w(\gamma,\delta)).$$

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, $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$ accepts a message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\mathsf{MPC}},1}$ from Z. Let $\mathsf{c} = \{c_{w,\gamma,\delta}\}_{w\in\{L:T\},\gamma,\delta\in\{0,1\}}$. $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$ outputs $(\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}[\sigma^0_{\mathsf{s}^0}], \widehat{\mu}^{(\widehat{\beta})})$, where $\mathsf{s}^0 = z_1\|\ldots\|z_N$. ($Z$'s output forms the auxiliary output of the generator $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}\|Z$.)

– Adversary Class $\mathcal{Q}^{\mathsf{Com},i}_{\mathsf{MPC}}$: It is the class of all PPT adversaries who must send the common random string $\mathsf{crs} = \{\mathsf{crs}_{w,\gamma,\delta}\}_{w\in\{L:T\},\gamma,\delta\in\{0,1\}}$ 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 $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$.

Next, we will show that the above classes decompose[27] to the corresponding classes for commitment opening and epsilon-transitions in the library, which already have R3PO schemes[28]. 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. *$(\mathcal{G}^{\mathsf{Com},i}_{\mathsf{MPC}}, \mathcal{Q}^{\mathsf{Com},i}_{\mathsf{MPC}})$, for every $i \in [n]$, decomposes into a library of commitment openings $(\mathcal{G}^{\mathsf{Com}}_{COMMIT}, \mathcal{Q}^{\mathsf{Com}}_{COMMIT})$ and epsilon-transitions $(\mathcal{G}^{\Sigma}_{\epsilon}, \mathcal{Q}^{\Sigma}_{\epsilon})$ (where, $\Sigma = \Sigma_{\mathsf{MPC}}$).*

Proof: We will show that $(\mathcal{G}^{\mathsf{Com},i}_{\mathsf{MPC}}, \mathcal{Q}^{\mathsf{Com},i}_{\mathsf{MPC}})$ decomposes into the library $\mathcal{L} = (\mathcal{L}_{w-L})_{w\in\{L:T\}}$, where

$$\mathcal{L}_{w-L} = \begin{cases} (\mathcal{G}^{\mathsf{Com}}_{\mathrm{COMMIT}}, \mathcal{Q}^{\mathsf{Com}}_{\mathrm{COMMIT}}) & \text{if } w \notin A_i, \\ (\mathcal{G}^{\Sigma}_{\epsilon}, \mathcal{Q}^{\Sigma}_{\epsilon}) & \text{otherwise.} \end{cases}$$

where $\Sigma = \Sigma_{\mathsf{MPC}}$. First, we will show the decomposition at partitions $w \notin A_i$. Therefore, for a generator $G \in \mathcal{G}^{\mathsf{Com},i}_{\mathsf{MPC}}$ and a partition $w \notin A_i$, we will describe J, Z so that $\forall Q \in \mathcal{Q}^{\mathsf{Com},i}_{\mathsf{MPC}}$, and all $(w-L)$-partial reach-extractors $\mathcal{E}$ for Q, the conditions given in Definition 17 hold.

– **Defining J.** Here is a description of the interaction between $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}, J, Q$ and $\mathcal{E}$:
  1. J reads $\mathsf{crs} = \{\mathsf{crs}_{w,\gamma,\delta}\}_{w\in\{L:T\},\gamma,\delta\in\{0,1\}}$ from Q and forwards it to $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$.

[27] Refer to Definition 17.
[28] These are also provided in the library.

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

$$\mathsf{s}^t = \mathsf{s}^{t-1}\|x_t.$$

where $\phi_{t+L} = (\cdot, u, v)$.
5. Set $\gamma = \mathsf{s}^{w-L-1}_u$ and $\delta = \mathsf{s}^{w-L-1}_v$, where $\phi_w = (i^* \neq i, u, v)$.
6. J samples a message function $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\mathsf{MPC}},w}$ and sends it to $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$.
7. J outputs $(\widehat{\pi}^{(\alpha)}_\sigma, \widehat{\mu}^{(\widehat{\beta})}_\sigma)$ as its auxilliary output, where $\sigma = \sigma^{w-L-1}_{\mathsf{s}^{w-L-1}}$ and $\widehat{\pi}^{(\alpha)}_\sigma$ is a one-step restriction (defined in Section 5) of the program $\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}$ (defined in Appendix E.2). Note that this one-step program only needs to have $\mathsf{crs}_{w,\gamma,\delta}$ and $c_{w,\gamma,\delta}$ hard-coded in it.

– **Defining Z.** Here is a description of the interaction between $\mathsf{H}^{\mathsf{Com}}_{\mathsf{COMMIT}}, Z, Q$ and $\mathcal{E}$:
1. Z reads $\mathsf{crs} = \{\mathsf{crs}_{w,\gamma,\delta}\}_{w\in\{L:T\},\gamma,\delta\in\{0,1\}}$ from Q.
2. Z samples $(z_i, \{\mathsf{G}_w\}_{w\in A_i}) \leftarrow \mathsf{pre}(1^\kappa, i, x_i)$, where x_i is its input.
3. For every $w \in A_i$, and $\gamma, \delta \in \{0,1\}$, Z generates

$$(c_{w,\gamma,\delta}, d_{w,\gamma,\delta}) \leftarrow \mathsf{Com.Commit}(\mathsf{crs}_{w,\gamma,\delta}, \mathsf{G}_w(\gamma,\delta)).$$

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.
6. Z receives $(a_Q, \Pi_{\Phi,\mathsf{x}}[\sigma^0_{\mathsf{s}^0}], X^*)$ from extractor $\mathcal{E}$. It parses $X^* = (x_t)_{t\in[T]}$, where x_t is a bit for all $t \in [T]$. Set $\mathsf{s}^0 = z_1\| \ldots \|z_n$. For every $t \in [T]$, Z does the following:

$$\mathsf{s}^t = \mathsf{s}^{t-1}\|x_t.$$

where $\phi_{t+L} = (\cdot, u, v)$.
7. Set $\gamma = \mathsf{s}^{w-L-1}_u$ and $\delta = \mathsf{s}^{w-L-1}_v$, where $\phi_w = (i^* \neq i, u, v)$.
8. It then provides $(c_{w,\gamma,\delta}, \mathsf{crs}_{w,\gamma,\delta})$ to $\mathsf{H}^{\mathsf{Com}}_{\mathsf{COMMIT}}$.
9. Z then samples $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\mathsf{COMMIT}},w}$ and sends $\widehat{\mu}^{(\widehat{\beta})}$ to $\mathsf{H}^{\mathsf{Com}}_{\mathsf{COMMIT}}$.
10. Z halts with output $(\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}[\sigma^0_{\mathsf{s}^0}], \widehat{\mu}^{(\widehat{\beta})})$, where $\mathsf{s}^0 = z_1\| \ldots \|z_N$ and $\mathsf{c} = \{c_{w,\gamma,\delta}\}_{w\in\{L:T\},\gamma,\delta\in\{0,1\}}$.

– **Indistinguishability check.** We need to show that the outputs of $\langle \mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}} \| J : Q\hat{|}\mathcal{E}\rangle$ and $\langle \mathsf{H}^{\mathsf{Com}}_{\mathsf{COMMIT}} \| Z : Q\hat{|}\mathcal{E}\rangle$ 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 σ of the non-reactive program $\widehat{\pi}^{(\alpha)}_\sigma$, 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 $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}, Q$ 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 $\sigma_1 := \sigma^{w-L-1}_{\mathsf{s}^{w-L-1}}$ and $\sigma_2 := \sigma^{w-L}_{\mathsf{s}^{w-L}}$, 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 $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_{\mathsf{MPC}},w}$ and sends it to $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$.
3. It outputs $(\widehat{\pi}^{(\alpha)}_{\sigma_1}, \widehat{\mu}^{(\widehat{\beta})}_{\sigma_1})$ as its auxilliary output, where $\widehat{\pi}^{(\alpha)}_{\sigma_1}$ is a one-step restriction of the program $\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}$ at state σ_1. Note that this one-step program only needs to have σ_1 and σ_2 hard-coded in it, since it is an ϵ-transition.

– **Defining Z.** In an interaction with $\mathsf{H}_\epsilon^\Sigma, Q$ and $\mathcal{E}$ (where $\Sigma = \Sigma_{\mathsf{MPC}}$), steps 1-6 are the same as in the description of Z for the previous case. Here are the following steps:
  1. Z sets $\sigma_1 := \sigma_{\mathsf{s}^{w-L-1}}^{w-L-1}$ and $\sigma_2 := \sigma_{\mathsf{s}^{w-L}}^{w-L}$, 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 $\widehat{\mu}^{(\widehat{\beta})} \in \widehat{\mathcal{M}}_{\mathcal{I}_\epsilon, w}$ and sends $((\sigma_1, \sigma_2), \widehat{\mu}^{(\widehat{\beta})})$ to $\mathsf{H}_\epsilon^\Sigma$.
  3. Z halts with output $(\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}[\sigma^0_{\mathsf{s}^0}], \widehat{\mu}^{(\widehat{\beta})})$, where $\mathsf{s}^0 = z_1 \| \ldots \| z_N$ and $\mathsf{c} = \{c_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$.
– **Indistinguishability check.** We need to show that the outputs of $\langle \mathsf{H}_{\mathsf{MPC}}^{\mathsf{Com},i} \| J : Q \hat{|} \mathcal{E} \rangle$ and $\langle \mathsf{H}_\epsilon^\Sigma \| Z : Q \hat{|} \mathcal{E} \rangle$ are indistinguishable, where $\Sigma = \Sigma_{\mathsf{MPC}}$. 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 σ of the non-reactive program $\widehat{\pi}^{(\alpha)}_\sigma$, the reach-restriction condition is satisfied.

□

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

Now, we give our construction of a 2-round MPC protocol in Figure 31. The construction is very similar to [30] except that the second round messages, consisting of multiple OT messages and a sequence of garbled circuits, have been replaced by an obfuscation of a transition program as described above. As a result, this leads to a much cleaner presentation. Correctness of this scheme follows from our description of Φ, Com and correctness of $\mathcal{O}^{\mathsf{Com},i}_{\mathsf{MPC}}$, for every $i \in [n]$. Also, let $\{L:T\}$ denote the set $\{L+1, \ldots, L+T\}$ in the following description.
For security, we prove the following lemma.

Lemma 20. *Assuming a conforming protocol* Φ *for a functionality, a UC-secure commitment scheme* Com*, and an R3PO scheme w.r.t.* $(\mathcal{G}^{\mathsf{Com},i}_{\mathsf{MPC}}, \mathcal{Q}^{\mathsf{Com},i}_{\mathsf{MPC}}, \mathring{\mathcal{P}}_{\mathsf{MPC}})$ *for every* $i \in [n]$*, there exists a secure 2-round* MPC *protocol for the same functionality.*

Proof: Let $\mathcal{A}$ 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 Sim that simulates the view of $\mathcal{A}$. Sim will internally use the simulator for malicious security of the underlying conforming protocol Sim_Φ, the extractors $(\mathcal{E}_1, \mathcal{E}_2)$ implied by binding property of Com, the simulator Sim_{Eq} implied by equivocal commitment property, the extractor $\mathcal{E}^{\mathsf{Com},i}_{\mathsf{MPC}}$ and the simulator $\mathsf{Sim}^{\mathsf{Com},i}_{\mathsf{MPC}}$ for security of $\mathcal{O}^{\mathsf{Com},i}_{\mathsf{MPC}}$, for every $i \in H$.

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

Simulating the interaction with $\mathcal{A}$: 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
$$\left(\mathsf{crs}_{w,\gamma,\delta}, c_{w,\gamma,\delta}, \{d^0_{w,\gamma,\delta}, d^1_{w,\gamma,\delta}\}\right) \leftarrow \mathsf{Sim}_{Eq}(1^\kappa).$$
  • For each $i \in [n] \setminus H,\ w \in A_i, \gamma, \delta \in \{0,1\}$, generate
$$(\mathsf{crs}_{w,\gamma,\delta}, \mathsf{state}_{w,\gamma,\delta}) \leftarrow \mathcal{E}_1(1^\kappa).$$
  • Output the common random string as $\{\mathsf{crs}_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma, \delta \in \{0,1\}}$.
– **Initialization:** Sim executes $\mathsf{Sim}_\Phi(1^\kappa)$ to obtain $\{z_i\}_{i \in H}$.

MPC from R3PO

Let Φ be a conforming protocol for the same functionality, Com be a UC-secure commitment scheme, and $\mathcal{O}_{\mathsf{MPC}}^{\mathsf{Com},i}$ be an R3PO scheme w.r.t. $(\mathcal{G}_{\mathsf{MPC}}^{\mathsf{Com},i}, \mathcal{Q}_{\mathsf{MPC}}^{\mathsf{Com},i}, \hat{\mathcal{P}}_{\mathsf{MPC}})$.

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

Round-1: Each party P_i does the following:

- Compute $(z_i, \{\mathsf{G}_w\}_{w\in A_i}) \leftarrow \mathsf{pre}(1^\kappa, i, x_i)$.
- For every $w \in A_i, \gamma, \delta \in \{0,1\}$

$$(c_{w,\gamma,\delta}, d_{w,\gamma,\delta}) \leftarrow \mathsf{Com.Commit}(\mathsf{crs}_{w,\gamma,\delta}, \mathsf{G}_w(\gamma,\delta)).$$

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

$$\mu_i^{(\beta)}(\sigma_{\mathsf{s}}^{w-L}) = \begin{cases} d_{w,\gamma,\delta} & \text{where } \phi_w = (i,u,v), \gamma = \mathsf{s}_u, \delta = \mathsf{s}_v, \\ \perp & \text{otherwise.} \end{cases}$$

  for all $w \in \{L:T\}$.
- Define $\alpha := \{\mathsf{G}_w\}_{w\in A_i}$.
- Set $\rho_i \leftarrow \mathcal{O}_{\mathsf{MPC}}^{\mathsf{Com},i}(\pi^{(\alpha)}_{\Phi,\mathsf{crs},\mathsf{c},i}[\sigma^0_{\mathsf{s}^0}], \mu_i^{(\beta)}, \mathcal{I}_{\mathsf{MPC}})$.
- Send ρ_i to every other party.

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

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

Fig. 31. Our MPC construction

- **Round-1 messages from Sim to $\mathcal{A}$:** For each $i \in H$, the simulator Sim sends $(z_i, \{c_{w,\gamma,\delta}\}_{w \in A_i, \gamma,\delta \in \{0,1\}})$ to $\mathcal{A}$ on behalf of honest party P_i.
- **Round-1 messages from $\mathcal{A}$ to Sim:** For each $i \in [n] \backslash H$, Sim receives the value $(z_i, \{c_{w,\gamma,\delta}\}_{w \in A_i, \gamma,\delta \in \{0,1\}})$ from $\mathcal{A}$ on behalf of corrupt party P_i. Next, for each $i \in [n] \setminus H, w \in A_i, \gamma, \delta \in \{0,1\}$, Sim extracts $b_{w,\gamma,\delta} := \mathcal{E}_2(\mathsf{state}_{w,\gamma,\delta}, c_{w,\gamma,\delta})$.
- **Completing the execution with Sim_{Φ}:** The interaction with Sim_{Φ} 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, Sim sends the bit $b_{w,\gamma,\delta}$ to Sim_{Φ}, where γ and δ are chosen appropriately based on the state vector maintained and ϕ_w. Let $\mathsf{Z} \in \{0,1\}^T$, where Z_t represents the bit sent in the t^{th} round of the computation phase of Φ, be the output of this execution. And let $\mathsf{s}^* \in \{0,1\}^{L+T}$ 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 $d_{w,\gamma,\delta} := d^{\mathsf{Z}_t}_{w,\gamma,\delta}$.
- **Round-2 messages from Sim to $\mathcal{A}$:** For each $i \in H$, the simulator Sim generates the second round message on behalf of P_i as follows:
  1. Set $\mathsf{c} := \{c_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma,\delta \in \{0,1\}}$.
  2. Run $\mathcal{E}^{\mathsf{Com},i}_{\mathsf{MPC}}$ internally by simulating an interaction between an $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$ and some Q; where these are described below.
  3. First, Q sends $\{\mathsf{crs}_{w,\gamma,\delta}\}_{w \in \{L:T\}, \gamma,\delta \in \{0,1\}}$ to $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$. The generator $\mathsf{H}^{\mathsf{Com},i}_{\mathsf{MPC}}$ 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 $\mathsf{x} = \mathsf{Z}$. Since $\mathcal{E}^{\mathsf{Com},i}_{\mathsf{MPC}}$ is allowed to look inside Q, it finally outputs $(a_Q, \Pi_{\Phi,\mathsf{x}}[\sigma^0_{\mathsf{s}^0}], \{\mathsf{x}\})$.
  4. For each $w \in A_i$ s.t. $\phi_w = (i, u, v)$, set $\gamma_w = \mathsf{s}^*_u$ and $\delta_w = \mathsf{s}^*_v$.
  5. Set $\rho_i \leftarrow \mathsf{Sim}^{\mathsf{Com},i}_{\mathsf{MPC}}(\pi^{(\cdot)}_{\Phi,\mathsf{crs},\mathsf{c},i}[\sigma^0_{\mathsf{s}^0}], \mu^{(\cdot)}_i, \mathcal{I}_{\mathsf{MPC}}, \Pi_{\Phi,\mathsf{x}}[\sigma^0_{\mathsf{s}^0}], \{\mathsf{x}, (\mathsf{x}_{w-L}, d_{w,\gamma_w,\delta_w})_{w \in A_i}\})$.
  6. Send ρ_i to $\mathcal{A}$.
- **Round-2 messages from $\mathcal{A}$ to Sim:** For every $i \in [n] \setminus H$, Sim obtains the second round message ρ_i from $\mathcal{A}$ 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.

- $\mathcal{H}_{\text{REAL}}$: This hybrid is the same as the real world execution.
- $\mathcal{H}'_{\text{REAL}}$: 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 $\mathcal{H}_0$ and $\mathcal{H}_{\text{REAL}}$ follows by the binding property of Com.
- $\mathcal{H}_0$: 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, Sim_{Eq} 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 Sim_{Eq} are hard-coded in the message function, inside the obfuscated program (the second-round message) for the corresponding honest party. Indistinguishability between hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ follows by the equivocal commitment security.
- $\mathcal{H}_i$ (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 $\mathcal{E}^{\mathsf{Com},i}_{\mathsf{MPC}}$ and $\mathsf{Sim}^{\mathsf{Com},i}_{\mathsf{MPC}}$ (just as in the simulation) to output the obfuscated program ρ_i.
  For this, the simulator Sim runs a mental execution of Φ 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 $\mathsf{Sim}^{\mathsf{Com},i}_{\mathsf{MPC}}$. Indistinguishability between two such consecutive hybrids follows from R3PO security of $\mathcal{O}^{\mathsf{Com},i}_{\mathsf{MPC}}$.
- $\mathcal{H}_{\text{IDEAL}}$: 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 Sim_{Φ}. Indistinguishability between $\mathcal{H}'_{\text{IDEAL}}$ and $\mathcal{H}_{\text{IDEAL}}$ follows from the malicious security of the underlying conforming protocol Φ. Finally, note that this is the same as the ideal world execution.

□