Succinct Witness Encryption for Batch Languages and Applications

Lalita Devadas[*] Abhishek Jain[†] Brent Waters[‡] David J. Wu[§]

Abstract

Witness encryption allows one to encrypt a message to an NP relation $\mathcal{R}$ and a statement x. The corresponding decryption key is any valid NP witness w. In a *succinct* witness encryption scheme, we require that the size of the ciphertext be sublinear in the size of the NP relation. Currently, all realizations of succinct witness encryption for NP rely on strong assumptions such as pseudorandom obfuscation, extractable witness encryption, or differing-inputs obfuscation. Notably, none of these notions are known from standard assumptions.

In this work, we consider a relaxation of succinct witness encryption for NP to the setting of *batch* NP. In this setting, one encrypts to an NP relation $\mathcal{R}$ together with K statements $x_1, \ldots, x_K$. In the basic version, one can decrypt if they have a witness $w_1, \ldots, w_K$ for all K statements. The succinctness requirement is that the size of the ciphertext should be *sublinear* in the number of instances K, but is allowed to grow with the size of the NP relation (i.e., the size of a *single* instance). More generally, we can also impose a (monotone) policy $P\colon \{0,1\}^K \to \{0,1\}$ over the K instances. In this case, decryption is possible only if there exists $w_1, \ldots, w_K$ such that $P(\mathcal{R}(x_1, w_1), \ldots, \mathcal{R}(x_K, w_K)) = 1$.

In this work, we initiate a systematic study of succinct witness encryption for batch languages. We start with two simple constructions that support CNF and DNF policies based on plain witness encryption in conjunction with a somewhere statistically sound batch argument for NP or a function-binding hash function. Then, using indistinguishability obfuscation, we show how to support policies that can be computed by read-once bounded-space Turing machines. The latter construction is in fact a unique witness map for monotone-policy batch NP, and as such, also gives a SNARG for monotone-policy batch NP where the size of the common reference string is *sublinear* in the number of instances.

Finally, we demonstrate some immediate applications of succinct witness encryption for batch languages. We construct new succinct computational secret sharing schemes for CNFs, DNFs, and weighted threshold policies. We also show how to build distributed monotone-policy encryption, a notion that generalizes recent trustless primitives like distributed broadcast encryption and threshold encryption with silent setup.

1 Introduction

In a witness encryption scheme (for NP) [GGSW13], a user can encrypt a message to an NP statement x. Anyone who knows an associated witness w can decrypt the ciphertext and recover the message. If the statement is false, then the message is computationally hidden. Witness encryption has been used to build a broad set of cryptographic primitives such as public-key encryption and generalizations such as identity-based encryption, attribute-based encryption, broadcast encryption [GGSW13, FWW23, WW24], oblivious transfer [BGI+17], laconic arguments [FNV17, BISW18, LMP24], and null obfuscation [WZ17, GKW17].

The witness size barrier. In witness encryption, the size of the ciphertext generally scales with the size of the NP witness. This is true for constructions based on indistinguishability obfuscation ($i\mathcal{O}$) [GGH+13], multilinear maps [GGSW13, GLW14], and lattice-based assumptions [CVW18, Tsa22, VWW22]. At a high level, we can view a witness encryption ciphertext as a program that takes an NP witness as input, checks if the witness is valid, and if so, outputs the message. Since this program takes the NP witness as input, the ciphertext size in existing constructions of witness encryption scale with the witness size (and more broadly, the size of the circuit computing the NP relation).

[*]MIT. Email: lali@mit.edu.

[†]NTT Research and Johns Hopkins University. Email: abhishek@cs.jhu.edu.

[‡]UT Austin and NTT Research. bwaters@cs.utexas.edu.

[§]UT Austin. Email: dwu4@cs.utexas.edu. Part of this work was done while visiting NTT Research.

A natural approach to overcome this witness-size dependency in the ciphertext size is to represent the NP relation as a Turing machine, where the description length of the machine is independent of the witness size. For instance, the ciphertext could be an obfuscated Turing machine that reads the witness and outputs the message if the witness is valid. While we can build witness encryption in this way using $i\mathcal{O}$ for Turing machines [BGL+15, CHJV15, KLW15, GS18], for general computations, the size of the obfuscated Turing machine still scales with the length of the input to the machine. Thus, these approaches still run into the witness size barrier. Using differing-inputs obfuscation [ABG+13] or extractable witness encryption [GKP+13], we can overcome this input-size barrier for $i\mathcal{O}$ for Turing machines. However, neither of these notions are currently known from standard assumptions.

Succinct witness encryption. Very recently, Branco et al. [BDJ+25] introduced the notion of *succinct* witness encryption where the size of the ciphertext is sublinear in the witness size. Moreover, they show how to overcome the witness size barrier and construct succinct witness encryption for general NP relations from a notion called *pseudorandom obfuscation*, which can be viewed as an ideal obfuscation for pseudorandom functions (i.e., the obfuscated program is computationally independent of the input program). While [BDJ+25] shows the feasibility of succinct witness encryption for NP, it relies on the new notion of pseudorandom obfuscation which we do not know how to instantiate using standard assumptions. The only such instantiations from [BDJ+25, AKY24] is based on learning with errors (LWE) together with the *private-coin* evasive LWE assumption. The work of [BDJ+25] also shows that pseudorandom obfuscation for arbitrary pseudorandom functions is impossible. As such, the security of their specific construction necessarily relies on a carefully-tailored formulation of private-coin evasive LWE. Moreover, a sequence of recent works [VWW22, BÜW24, BDJ+25, AMYY25, HJL25, HHY25] have raised significant questions on the plausibility of private-coin evasive LWE. Thus, a natural question to ask is whether we can build succinct witness encryption from standard cryptographic assumptions.

1.1 Our Results

In this work, we conduct a systematic study of succinct witness encryption for *subclasses* of NP. Specifically, we consider batch languages, where instead of encrypting to a single NP statement, one instead encrypts to a batch of K statements $x_1, \ldots, x_K$. Decryption succeeds if one knows valid witnesses w_i for some subset of the statements. The succinctness requirement is that the ciphertext scales *sublinearly* with the total number of statements K, but we do allow the ciphertext to scale with the size of a *single* witness. In some sense, this is the same type of relaxation considered in the study of batch arguments (BARGs) for NP [BHK17], where the goal is to give a succinct proof of a batch of statements with a proof whose size scales sublinearly with the number of instances.

The basic version of our notion requires the decrypter to know a witness for *every* statement in order to decrypt. However, we can consider more general decryption policies such as a threshold policy where one needs to know t-out-of-n witnesses in order to decrypt. Most generally, we consider monotone policies where the encryption algorithm takes the statements $x_1, \ldots, x_K$ together with a monotone policy $P \colon \{0,1\}^K \to \{0,1\}$, and decryption is successful only if there exist witnesses $w_1, \ldots, w_K$ such that $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$. Here C is the circuit that implements the associated NP relation. This setting is the analog of monotone-policy batch arguments [BBK+23, NWW24, NWW25] in the setting of succinct witness encryption.

The recent work of [ADM+24] considers a conceptually-similar notion of succinct witness encryption for batch languages, but their work has two key restrictions. First they focus on witness encryption for a *specific* cryptographic relation (namely, knowledge of a signature) and second, they only consider threshold policies. Their work gives a construction from $i\mathcal{O}$ for Turing machines. In this work, our focus is on constructing succinct witness encryption for batch NP and we also explore broader policy classes. We give constructions for simple policy families using plain witness encryption and constructions for more expressive policies (e.g., monotone policies that can be computed by read-once bounded-space Turing machines) using $i\mathcal{O}$.

Constructions of succinct witness encryption. In this work, we provide three constructions of succinct witness encryption for batch languages for different policy families. Our first two constructions support CNF and DNF policies and are based on (plain) witness encryption. Our third construction relies on $i\mathcal{O}$ and supports any monotone policy that can be computed by a read-once log-space Turing machine as well as policies like weighted thresholds (with $\mathsf{poly}(\lambda)$-bit weights). We summarize the key features of our constructions below:

- **Conjunctions of local predicates.** Our first construction of succinct witness encryption supports conjunctions of local monotone predicates. Specifically, consider a policy $P\colon \{0,1\}^K \to \{0,1\}$ of the form

$$P(\beta_1, \ldots, \beta_K) \coloneqq P_1(\vec{\beta}_{S_1}) \wedge \cdots \wedge P_c(\vec{\beta}_{S_c}),$$

where each P_i is a monotone predicate that depends on a subset $S_i \subseteq [K]$ of the inputs, and we write $\vec{\beta}_{S_i}$ to denote the subset of $\beta_1, \ldots, \beta_K$ indexed by S_i. An encryption of a message μ with respect to a Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, policy P, and instances $x_1, \ldots, x_K \in \{0,1\}^n$ has size $|\mu| + \mathsf{poly}(\lambda, |C|, s, \log c)$, where s is the size of the largest predicate P_i. For instance, when each predicate P_i depends on a constant number of input variables, then $s = O(1)$ and the size of the ciphertext is $\mathsf{poly}(\lambda, |C|, \log |P|)$, which scales with the size of a *single* instance and only polylogarithmically with the number of instances. This captures important special cases where the predicate P is a conjunction or a 3-CNF formula. We construct our succinct witness encryption scheme for conjunctions of local predicates from plain witness encryption and a *somewhere-statistically-sound* batch argument for NP (e.g., [WW22]; see Section 1.2 for a description of the somewhere-statistically-soundness requirement).

- **Disjunction of local predicates.** We then consider the dual notion of succinct witness encryption for *disjunctions* of local monotone predicates.[1] Specifically, our construction supports policies of the form

$$P(\beta_1, \ldots, \beta_K) \coloneqq P_1(\vec{\beta}_{S_1}) \vee \cdots \vee P_c(\vec{\beta}_{S_c}),$$

where each P_i is a monotone predicate that depends on a subset $S_i \subseteq [K]$ of the inputs. An encryption of a message μ with respect to a Boolean relation C, policy P, and instances $x_1, \ldots, x_K$ has size $|\mu| + \mathsf{poly}(\lambda, |C|, s, \log c)$, where s is the size of the largest predicate P_i. Once again, the ciphertext size scales with the size of a single instance and the size of the largest predicate, and only polylogarithmically with the number of predicates. This constructions captures notions like DNF formulas as a special case. We construct our succinct witness encryption scheme for disjunctions of local predicates from plain witness encryption and a function-binding hash function [FWW23]; the latter can be built from any leveled homomorphic encryption scheme.

- **Read-once bounded-space Turing machine policies from $i\mathcal{O}$.** Our final construction supports monotone policies that can be computed by a read-once bounded-space Turing machine. Specifically, suppose $P\colon \{0,1\}^K \to \{0,1\}$ is a policy that can be computed by a read-once Turing machine with S-bits of space. Then an encryption of a message μ with respect to a Boolean relation C, policy P, and instances $x_1, \ldots, x_K$ has size $|\mu| + \mathsf{poly}(\lambda, |C|, 2^S)$. Note that the size of the ciphertext scales *exponentially* with the space usage, so this scheme is tailored for policies that can be computed by read-once Turing machines with $O(\log \lambda)$-bits of space. For instance, this captures policies such as threshold policies. Technically, our scheme is more general and can support monotone policies computable by read-once Turing machines where the set of unreachable states has a compact description. For instance, this allows our scheme to also support *weighted* threshold policies with λ-bit weights (i.e., weights with magnitude 2^λ). Our construction here relies on $i\mathcal{O}$ for circuits together with somewhere-statistically-binding (SSB) hash functions [HW15].

Succinct unique witness maps for monotone-policy batch NP. Our $i\mathcal{O}$-based construction for read-once bounded-space Turing machines is more general and in particular, gives a succinct *unique witness map* [CPW20] for monotone-policy batch NP with respect to policies computable by read-once bounded-space Turing machines. A unique witness map for an NP relation deterministically maps all witness for an NP statement onto a single "canonical" witness. There is a public verification algorithm that decides whether a candidate witness is the canonical witness for a statement. The soundness requirement is that for a false statement, an efficient adversary cannot produce a witness that satisfies the verification relation. In this work, we show how to construct a *succinct* unique witness map for monotone-policy batch NP where the policy can be computed by a read-once bounded-space Turing machine. The succinctness requirement is that the size of the common reference string (CRS) for the unique witness map is

[1]Technically, this construction only supports "trapdoor NP relations" (see Definition 4.11) where we assume there is an efficient algorithm that can decide the instance with the help of a trapdoor. This is a subclass of NP, but one that suffices for the cryptographic applications of succinct witness encryption we consider in this work (Section 6).

sublinear in K and and the size of the canonical witness for K instances has size $\text{poly}(\lambda, \log K)$. Notably, the size of the canonical witness is *fully* succinct (independent of the size of the NP relation). A succinct unique witness map for monotone-policy batch NP with policy family $\mathcal{P}$ immediately yields the following:

- A succinct witness encryption scheme for batch NP with policy family $\mathcal{P}$ (see Remark 5.3). This follows via the same "or-proof with hard-core predicate" transformation in [CPW20] (who described the implication from a unique witness map for NP to witness encryption for NP).
- A succinct non-interactive argument (SNARG) for monotone-policy batch NP with policy family $\mathcal{P}$ where the size of the CRS is sublinear in the number of instances (see Remark 5.2). Previously, SNARGs for batch NP with a sublinear-size CRS were only known for conjunction policies [GSWW22, DWW24]. Our work gives the first construction that supports policies like weighted thresholds. Even *without* full succinctness (i.e., monotone-policy BARGs where the proof size can scale with the size of the NP relation and just needs to be sublinear in the number of instances), previous constructions [BBK+23, NWW24, NWW25] also required a CRS whose size scales linearly with the number of instances.

Applications of succinct witness encryption for batch languages. We then highlight two immediate applications of succinct witness encryption for batch languages. The first is to succinct computational secret sharing [ABI+23] and the second is a notion we call distributed monotone-policy encryption, which is a generalization of notions like distributed broadcast encryption [WQZD10, BZ14] and threshold encryption with silent setup [GKPW24, ADM+24]:

Succinct computational secret sharing. In a succinct computational secret sharing scheme [ABI+23], a dealer can share a secret with K parties with respect to a monotone access policy P. Thereafter, any subset of parties that satisfies the access policy can come together and reconstruct the secret while the shares of any unauthorized set of users should computationally hide the message. The succinctness requirement is that the size of each share should be sublinear in the number of parties and the description size of P. A succinct witness encryption scheme for batch NP with policy family $\mathcal{P}$ immediately implies a computational secret sharing scheme for the same family $\mathcal{P}$. Thus, our work gives a succinct computational secret sharing scheme for monotone CNF and DNF formulas from plain witness encryption (together with the LWE assumption) as well as for monotone policies computable by read-once log-space Turing machines from $i\mathcal{O}$ (and SSB hash functions). Our construction for DNFs additionally assumes the parties share a long, but reusable common random string, which can be compressed using a random oracle.

Previously, the works of [HIJ+16, BCG+19, ASY22] show how to use pseudorandom correlation generators based on $i\mathcal{O}$ to build succinct computational secret sharing for general monotone policies in a setting where parties have access to a long, reusable common random string (or alternatively, in the random oracle model). In the plain model, the work of [ABI+23] shows how to construct succinct computational secret sharing for CNFs using either RSA or $i\mathcal{O}$. For DNF policies, the work of [ABI+23] give a construction with a *public* share whose size is *linear* in the policy size. Our constructions improve upon these approaches as follows:

- For k-CNF policies (with constant k), we give a construction based on witness encryption in the plain model. This is the first construction that does not use $i\mathcal{O}$ or the RSA assumption.
- For k-DNF policies (for any constant k), we obtain a construction from witness encryption in the random oracle model. Previous constructions either relied on $i\mathcal{O}$ in the random oracle model [HIJ+16, BCG+19, ASY22] or on one-way functions but with a long (message-dependent) public share [ABI+23]. Strictly speaking, the construction of [ABI+23] does not satisfy the standard succinctness requirement for succinct computational secret sharing which requires all shares to be sublinear in the size of the policy.
- Using $i\mathcal{O}$, we obtain the first succinct computational secret sharing scheme for weighted threshold policies where the share size is $\text{poly}(\lambda, \log W)$ in the plain model and W is the magnitude of the weights. Previously, this was only known from $i\mathcal{O}$ in the random oracle model [HIJ+16, BCG+19, ASY22]. Technically, our construction supports any policy family that can be computed by a read-once, log-space Turing machine.[2]

[2]The actual class we capture is more general (see Section 1.2), and includes weighted thresholds (which is not captured generically by a read-once, log-space Turing machine when the weights are $\omega(\log \lambda)$-bits).

Distributed monotone-policy encryption. Distributed monotone-policy encryption is a generalization of distributed broadcast encryption [WQZD10, BZ14] and threshold encryption with silent setup [GKPW24, ADM+24]. Specifically, in a threshold encryption scheme with silent setup, individual users each choose a public/secret key-pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. Each user publishes their public key pk_i in a public key directory. Later on, an encrypter can select any subset of public keys $\{\mathsf{pk}_i\}_{i \in S}$ and a threshold $t \leq |S|$ and encrypt a message μ to this set. The guarantee is that any set of at least t users in the set S can decrypt and recover the message μ, while any subset of less than t users are unable to do so. Moreover, the size of the ciphertext should be succinct (scaling independently of the size of the number of public keys in S). Distributed broadcast encryption is the special case where the threshold $t = 1$ (i.e., any individual user in the set S can decrypt).

Existing constructions of distributed monotone-policy encryption have thus far been limited to simple policies: threshold encryption [GKPW24, ADM+24] or distributed broadcast encryption [BZ14, FWW23, KMW23, CW24, CHW25, WW25b]. Using succinct witness encryption for batch languages, we immediately obtain the first constructions that can support more general policies including CNFs and DNFs from witness encryption, and weighted thresholds and more from $i\mathcal{O}$.

1.2 Technical Overview

We begin with an overview of our constructions and show how to use succinct witness encryption for batch languages to realize other cryptographic primitives.

Succinct witness encryption for conjunctions. As a warm-up, we show how to construct succinct witness encryption for conjunctions from plain witness encryption and a *somewhere-statistically-sound* batch argument (BARG). In a somewhere-statistically-sound BARG for NP [WW22], a prover can prove a batch of K NP statements $x_1, \ldots, x_K$ with a proof whose size scales sublinearly with K. Moreover, the public parameters for the BARG can be programmed with a special index i such that the BARG is *statistically* sound on index i. In other words, when the CRS is binding on index i, with overwhelming probability over the choice of i, there does not exist[3] an accepting proof for any batch of instances $(x_1, \ldots, x_K)$ where x_i is false. The special index i is computationally hidden from the view of the prover. In some sense, a BARG proof π can be viewed as a "compressed" witness for the tuple $(x_1, \ldots, x_K)$. We leverage this to construct a succinct witness encryption scheme for conjunctions:

- Suppose we want to encrypt a message μ to instances $(x_1, \ldots, x_K)$ for an NP relation $\mathcal{R}$. We want decryption to be possible only if the user knows a witness for *all* K instances.
- We compress the instances by hashing them. Let h be a hash of $(x_1, \ldots, x_K)$. We assume that the hash function supports local openings (i.e., it is possible to open h to any x_i with an opening of size $\mathrm{poly}(\lambda, \log K, |x_i|)$).
- The ciphertext is a witness encryption ciphertext for the NP relation that verifies a BARG proof on K instances, where the witness for instance i consists of a purported statement x_i, a local opening of x_i with respect to the hash h, and a witness w_i for x_i. The BARG relation would check the validity of the local opening to x_i and then check that $\mathcal{R}(x_i, w_i) = 1$.

Correctness follows immediately from completeness of the BARG. Succinctness follows from succinctness of the BARG and the hash function. Security relies on security of the underlying witness encryption scheme and somewhere-statistical-soundness of the BARG. Specifically, we show that if any individual instance x_i is false, then there does not *exist* a BARG proof that verifies (with respect to the hash digest h). To argue this, we require that the hash function be somewhere statistically binding [HW15] (namely, the hash key can be programmed at a hidden index i such that any hash h can only be opened to one particular value at position i) and the BARG to be somewhere statistically sound. Then, we proceed as follows:

- Take any $(x_1, \ldots, x_K)$ where x_i is false. First, we program the hash function to be statistically binding at index i and similarly, we program the BARG to be somewhere statistically sound on index i.

[3] Other BARG constructions such as [CJJ21b, CGJ+23] only ensure somewhere *computational* soundness which stipulates that such proofs are hard to find for an efficient adversary.

- Let h be an SSB hash of $(x_1, \ldots, x_K)$. The encryption algorithm encrypts with respect to the honestly-computed hash h. Since h is statistically binding on index i, there does not exist a valid opening σ_i of h to any string $x'_i \neq x_i$ at index i.
- Now, the i^{th} instance of the BARG is false, since it consists of either an invalid local opening of h to some $x'_i \neq x_i$, or a valid opening of h to the false statement x_i. Since the BARG is statistically sound on index i, there does not exist an accepting BARG proof.
- Thus, the plain witness encryption scheme is being applied to a false statement (since a witness would be an accepting BARG proof), so we can appeal to semantic security of the plain witness encryption scheme.

This immediately gives a succinct witness encryption for conjunctions. The construction naturally generalizes to CNFs and conjunctions of local predicates, and we give the details in Section 4.1.

Succinct witness encryption for disjunctions. Next, we turn to the setting of disjunctions. Here, we show how to combine plain witness encryption with a *function-binding hash function* [FWW23] to obtain a succinct witness encryption scheme for disjunctions. Essentially, the function-binding hash function plays the role of the index BARG for compressing the instances. Function-binding hash function (for disjunctions) generalize SSB hash functions by (statistically) binding to a function of the input. Specifically, they have the following syntax:

- Like SSB hash functions, a user can use the hash key to compute a hash h of an input $(x_1, \ldots, x_K)$ and produce a succinct local opening of x_i with respect to the hash.
- Next, the hash key can be sampled to be function binding for a specific function g. Like [FWW23], we consider disjunctions of the form
$$f(x_1, \ldots, x_K) \coloneqq \bigvee_{i \in [K]} g(x_i),$$
where g is some predicate. The function binding property now states that if h is a hash of $(x_1^*, \ldots, x_n^*)$ where $f(x_1^*, \ldots, x_n^*) = 0$, then there does *not* exist an opening to any $\hat{x}_{i^*}$ at any index $i^* \in [K]$ where $g(\hat{x}_{i^*}) = 1$. Moreover, the hash key *hides* the associated function f.

To construct a succinct witness encryption scheme for disjunctions, we now proceed as follows:

- Suppose we want to encrypt μ to instances $(x_1, \ldots, x_K)$ with respect to an NP relation $\mathcal{R}$. We want to support decryption if the user know a witness w_i for any instance x_i.
- The encrypter starts by computing a hash h of $(x_1, \ldots, x_K)$ using the function-binding hash function. Then, the encrypter prepares a witness encryption of the message μ with respect to the NP relation that takes as input a purported statement x_i, an opening of x_i with respect to the hash h, and a witness w_i for x_i. The witness encryption simply checks that that the local opening is valid and that $\mathcal{R}(x_i, w_i) = 1$. The size of the ciphertext grows with the size of the verification circuit for the function-binding hash function and the size of the NP relation $|\mathcal{R}|$.

Security of the above construction relies on the function-binding property of the hash function. Here, we will need to additionally assume that the NP relation $\mathcal{R}$ is a "trapdoor NP relation:" namely, given some trapdoor information, there is an efficient algorithm that decides membership in the NP language. Many cryptographic languages are trapdoor NP relations (e.g., the set of strings that are encryptions of 1 under a public-key encryption scheme). As we discuss later (see also Section 6), when using witness encryption to build other cryptographic primitives, it is typically applied to a cryptographic language. Indeed, both of the applications we consider in this work (to succinct computational secret sharing and to distributed monotone-policy encryption) can be instantiated from succinct witness encryption for batch trapdoor NP relations.

To prove security of this construction, we start by fixing a false instance $(x_1, \ldots, x_K)$. Let C_{td} be the trapdoor-decision algorithm associated with $\mathcal{R}$. Namely, $C_{\mathsf{td}}(x) = 1$ if and only if there exists a witness w where $\mathcal{R}(x, w) = 1$. Since $(x_1, \ldots, x_K)$ is false, this means $C_{\mathsf{td}}(x_i) = 0$ for all $i \in [K]$. Suppose we now program the function-binding hash function to be function-binding on the function $f(x_1, \ldots, x_K) = \bigvee_{i \in [K]} C_{\mathsf{td}}(x_i)$. Now, if h is a hash of $(x_1, \ldots, x_K)$, the function-binding property says there cannot be *any* opening of h to a statement $\tilde{x}_i$ where $C_{\mathsf{td}}(\tilde{x}_i) = 1$. Correspondingly, the only openings that exist for h are to statements $\tilde{x}_i$ where $\mathcal{R}(\tilde{x}_i, \tilde{w}_i) = 0$ for all $\tilde{w}_i$. Thus, there does not exist a valid witness for the witness encryption relation, and security follows from security of the witness encryption scheme.

Similar to our construction for conjunctions, this scheme readily generalizes to support disjunctions of arbitrary local monotone predicates. We provide the full details and formal analysis in Section 4.2.

Unique witness maps for read-once Turing machines. Our main construction is a unique witness map based on $i\mathcal{O}$ for monotone-policy batch NP that supports any policy that can be computed by a read-once bounded-space Turing machine. For our applications, we require two types of succinctness: the common reference string for the unique witness map must be sublinear in the number of instances K, and the size of the canonical witness for a batch of statements $(x_1, \ldots, x_K)$ should be $\mathsf{poly}(\lambda, \log K)$, and be essentially independent of the size of the NP relation or the size of the associated policy. In our construction, the size of the public parameters will scale with 2^S where S is the space usage of the Turing machine. As we see later, the size actually scales with the description length of the set of unreachable states associated with the policy, which for certain policy families, is much smaller than 2^S. An important example of this is the class of weighted thresholds (see Remark 5.7).

The common reference string (CRS) for our unique witness map consists of two (obfuscated) programs: (1) a prover program that is used to map witnesses for a batch of K statements together with an associated policy onto a canonical witness; and (2) a verification program for verifying a canonical witness. Since we require the CRS size to be sublinear in K, we cannot publish an obfuscated program that reads all K instances at once. Instead, similar to [GSWW22, DWW24] (which consider SNARGs for batch NP with respect to conjunction policies), the prover program reads one statement/witness at a time. After reading each statement/witness, it then outputs a representation of its internal state together with a signature on it. The evaluator then invokes the prover program on the next instance. More concretely, the programs behave as follows:

- First, we assume the NP relation (expressed as a Boolean circuit C that computes the NP relation) is hard-coded in the prover and verification programs.
- We model a policy $P \colon \{0,1\}^K \to \{0,1\}$ as a read-once Turing machine with S bits of space. Concretely, we represent it as a tuple $(\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$, where $\mathsf{Step}_i \colon \{0,1\}^S \times \{0,1\} \to \{0,1\}^S$ is a Boolean circuit that takes the current configuration and the next bit of the input and outputs the next state, $c_{\mathsf{init}} \in \{0,1\}^S$ is the initial configuration, and $c_{\mathsf{acc}} \in \{0,1\}^S$ is the accepting configuration. Let h be a hash on the tuple $((x_1, \mathsf{Step}_1), \ldots, (x_K, \mathsf{Step}_K))$.[4]
- To compute the canonical witness, the evaluator runs the obfuscated proving program K times. On the i^{th} iteration, the evaluator passes in the hash h, the instance x_i, the step function Step_i together with an opening for (x_i, Step_i) relative to h, the associated witness w_i, the configuration c_{i-1} from the first $i-1$ steps of the evaluation (with $c_0 \coloneqq c_{\mathsf{init}}$), and a signature σ_{i-1} on $(\mathsf{h}, i-1, c_{i-1})$.
- If the signature σ_{i-1} is valid, then the proving program computes the bit $\beta_i = C(x_i, w_i)$ and the updated state of the Turing machine $c_i = \mathsf{Step}_i(c_{i-1}, \beta_i)$. It outputs a signature σ_i on (h, i, c_i).
- The canonical witness for the batch of K statements is a signature on $(\mathsf{h}, K, c_{\mathsf{acc}})$, where c_{acc} is the accepting configuration.
- The verification algorithm calls the obfuscated verification program which checks whether it was given a valid signature on $(\mathsf{h}, K, c_{\mathsf{acc}})$ or not.

For any tuple of statements $(x_1, \ldots, x_K)$ and policy P, the canonical witness is a (deterministic) signature on $(\mathsf{h}, K, c_{\mathsf{acc}})$. This depends solely on the statements and the policy, and is *independent* of the witnesses used to derive the signature. Thus, this constructive gives a unique witness map for batch instances.

Proving security of this construction is very delicate. Recall that the security requirement for a unique witness map says that an efficient adversary cannot come up with a proof that passes verification for any tuple $(x_1, \ldots, x_K)$ that does not satisfy the policy P. The general approach we take combines the chaining approach from [GSWW22, DWW24]

[4]In the actual construction (Construction 5.8), we separate these into two separate hashes in order to support a local evaluation property that is useful for applications (see Remark 5.11). For ease of exposition in this overview, we describe things using a single hash.

with the pebbling arguments from [GPSZ17, GS18]. Notably, the latter was used for analyzing $i\mathcal{O}$ for Turing machines. We highlight some of the key features of our reduction and refer to Section 5 for the formal description:

- First, we consider selective security where the statements $(x_1, \ldots, x_K)$ and the monotone policy P are fixed in advance. Now, for each $i \in [K]$, we define the set S_i of reachable configurations after i steps of P. Specifically, let $S_0 = \{c_{\mathsf{init}}\}$ be the initial configuration. Then, for each $i \in [K]$, S_i contains all c_i where $c_i = \mathsf{Step}(c_{i-1}, 0)$ and $c_{i-1} \in S_{i-1}$ (i.e., the states that are reachable by starting from a reachable state after $i-1$ steps and then reading a 0). In addition, if the i^{th} statement is true, then S_i also contains all c_i where $c_i = \mathsf{Step}(c_{i-1}, 1)$ where $c_{i-1} \in S_{i-1}$ (i.e., the states that are reachable from a reachable state after i steps and then reading a 1).

- Let h^* be the hash on $((x_1, \mathsf{Step}_1) \ldots, (x_K, \mathsf{Step}_K))$. Our general strategy in the security proof is to propagate the following invariant: at step i of the computation, the only possible signatures on tuples of the form (h^*, i, c_i) are those where $c_i \in S_i$. Namely, signatures should only exist on *reachable* configurations. We show that if the invariant holds for an index i, then using punctured programming techniques [SW14], we can establish the invariant for index $i+1$. To carry out this step, we need to embed within the obfuscated program a description of the reachable states after the first i states (i.e., a description of the set S_i). For this reason, the size of the obfuscated program in our construction grows exponentially with the space of the Turing machine, limiting us to log-space computations. However, in settings where the the sets S_i have a compact description, then the size of the programs only scales with the size of the compact description of S_i. This captures important policies families such as weighted threshold policies (see Remark 5.7).

- The next challenge comes from the need to "unpuncture." Specifically, the way we establish our invariant that signatures do not exist on any unreachable configuration is we "puncture" away the ability to generate signatures on inputs of the form (h^*, i, c_i) for all $c_i \notin S_i$. If we have to hard-code all of these inputs into the program across all indices $i \in [K]$, then the size of the program scales linearly with the number of instances, which is precisely what we want to avoid. Thus, once we have established the invariant on an index i, we need a way to unpuncture previous points. We can model this unpuncturing step as a pebbling game [GPSZ17, GS18], which provides a way to propagate our invariant while ensuring that at any point in time, we only need to program in the sets S_i for at most $O(\log K)$ indices. We refer to the proof of Theorem 5.13 for the full details.

- A second challenge that arises in our analysis is within the propagation step itself. Recall that on step i, our goal is to puncture away all signatures on inputs of the form (h^*, i, c_i) where $c_i \notin S_i$. A natural approach would be to step through each of the 2^S possible configurations one-by-one and leverage punctured programming (and indeed, this is our overall proof strategy). This would result in 2^S hybrids in total, and if done naïvely, would lead to signatures whose size now scales with the space-bound S. Once again, this would no longer meet our succinctness requirements. Thus, we need to design our hybrids so that the dependence on the space usage S is entirely absorbed by the programs in the common reference string and not in the size of the output signatures. To support this, we use the randomization techniques developed recently in the context of constructing and batching adaptively-sound SNARGs for NP from $i\mathcal{O}$ [DWW24, WW25a].

Taken together, we obtain a succinct witness map for batch languages from $i\mathcal{O}$ and SSB hash functions. The size of the CRS scales with the description length of the set of reachable states for the Turing machine. Note that if we directly use $i\mathcal{O}$ for Turing machines to build a unique witness map (i.e., read all K statements/witnesses and output a signature if the policy is satisfied), then the size of the CRS grows linearly with the number of instances K (because the size of the obfuscated program in existing constructions of $i\mathcal{O}$ for Turing machines from standard assumptions [BGL+15, CHJV15, KLW15, GS18] all grow with the input length). These approaches would in turn not suffice for either of our applications to succinct witness encryption or for SNARGs for monotone-policy batch NP with a sublinear-size CRS. Our approach shows how to overcome this input-size barrier for a particular class of computations.

Application to succinct computational secret sharing. A succinct witness encryption scheme for a policy $\mathcal{P}$ immediately gives a succinct computational secret sharing scheme for $\mathcal{P}$. The construction is straightforward:

- Let pk be the public-key for a public-key encryption scheme. Suppose there are K parties in the system. Each party's share consists of a pair (ct_i, r_i) where $\mathsf{ct}_i = \mathsf{Encrypt}(\mathsf{pk}, 1; r)$ is encryption of 1 under pk using randomness r_i.

- Let $\mathcal{R}_{\mathsf{pk}}(\mathsf{ct}, r)$ be an NP relation that outputs 1 if $\mathsf{ct} = \mathsf{Encrypt}(\mathsf{pk}, 1; r)$. To share a message μ with respect to a policy P, the dealer publishes a witness encryption ciphertext with message μ, statement $(\mathsf{ct}_1, \dots, \mathsf{ct}_K)$, policy P, and $\mathcal{R}_{\mathsf{pk}}$ as the associated NP relation.

Both the parties' individual shares as well as the public information are succinct (the latter by succinctness of the witness encryption scheme). Moreover, any set of users that satisfies the policy is able to decrypt and recover μ.[5] Security follows readily from security of the succinct witness encryption scheme. We refer to Section 6.1 for the full details. An appealing feature of our construction is that the party's individual shares are *reusable*. Whenever the dealer wants to share a new message, it only needs to publish a short *public* share (i.e., the witness encryption ciphertext).

Application to distributed monotone-policy encryption. Our second application is to distributed monotone-policy encryption. In this notion, users generate their own public keys pk and post them to a public-key directory. Thereafter, one can encrypt to a collection of public keys $\{\mathsf{pk}_i\}_{i \in S}$ together with a monotone policy P (on the set S of public keys). Any authorized set of users can pool their decryption keys to decrypt and learn the message. We can construct a distributed monotone-policy encryption scheme in the same manner as our succinct computational secret sharing construction. Namely, the public parameters for the monotone-policy encryption scheme will be the public key pk for a public-key encryption scheme. Each user's public key will be an encryption of 1 under pk and their secret key is the associated encryption randomness. An encryption of a message μ to a set of public keys $\{\mathsf{pk}_i\}_{i \in S}$ and access policy P is precisely a succinct witness encryption of μ where the statement corresponds to $\{\mathsf{pk}_i\}_{i \in S}$ and the policy corresponds to P. Correctness and security follow as in the case of succinct computational secret sharing.

In the basic version described here, users have to share their decryption keys in order to decrypt. We can also consider a stronger model where instead of users exchanging secret keys to decrypt they instead publish a one-time partial decryption of the message. This is standard in settings like multi-key homomorphic encryption [MW16] or threshold encryption with silent setup [GKPW24, ADM⁺24]. Using similar techniques as in [GKPW24, ADM⁺24], we can easily extend our construction to support this. Namely, we let each user's long-term secret key be a signing key for a digital signature scheme. Their public key is the associated verification key. Next, we associate a (random) tag with each ciphertext. Instead of checking for possession of the associated decryption key, the witness encryption scheme now checks that the decrypter provides a valid signature for a set of users that satisfy the access policy. The signatures now serves as the "decryption key." Security of the signature scheme ensures that the decryption hint for one ciphertext does not help break semantic security of an unrelated ciphertext. We provide the full details in Appendix B.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n] := \{1, \dots, n\}$. We say a function $f(\lambda)$ is negligible if $f = o(\lambda^{-c})$ for all $c \in \mathbb{N}$, and write $\mathsf{negl}(\lambda)$ to denote a negligible function. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We write $\mathsf{poly}(\lambda)$ to denote a fixed function that is bounded by some polynomial in λ. We model an efficient non-uniform algorithm $\mathcal{A}$ as a pair of algorithms $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ where $\mathcal{A}_0$ is a (possibly unbounded) algorithm that takes as input 1^λ and outputs an advice string ρ_λ of polynomial length, and algorithm $\mathcal{A}_1$ is an efficient algorithm. The output of $\mathcal{A}$ on an input $x \in \{0,1\}^\lambda$ is defined as first computing the advice string $\rho_\lambda \leftarrow \mathcal{A}_0(1^\lambda)$ and then taking the output to be $\mathcal{A}_1(\rho_\lambda, x)$. Throughout this work, we will consider efficient non-uniform adversaries.

Witness encryption. We now recall the notion of a witness encryption scheme for NP [GGSW13].

Definition 2.1 (Witness Encryption [GGSW13, adapted])**.** Let $\mathcal{M}$ be a message space. A witness encryption scheme Π_{WE} for NP with message space $\mathcal{M}$ is a pair of efficient algorithms $\Pi_{\mathsf{WE}} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Encrypt}(1^\lambda, C, \mu) \to \mathsf{ct}$: On input the security parameter $\lambda \in \mathbb{N}$, a circuit $C \colon \{0,1\}^h \to \{0,1\}$, and a message $\mu \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.

[5]Technically, this requires that decryption is possible even if the users do *not* know the full statement (i.e., the public keys of all users in the system). As we discuss in Sections 4 and 5, the schemes in this work support this "local" decryption property.

- $\mathsf{Decrypt}(\mathsf{ct}, C, w) \to \mu$: On input a ciphertext ct, a Boolean relation $C\colon \{0,1\}^h \to \{0,1\}$, a witness $w \in \{0,1\}^h$, the decryption algorithm outputs a message μ.

Moreover, we require that $(\mathsf{Encrypt}, \mathsf{Decrypt})$ satisfy the following two properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$, all Boolean circuits $C\colon \{0,1\}^h \to \{0,1\}$, all witnesses $w \in \{0,1\}^h$ where $C(w) = 1$, and all messages $\mu \in \mathcal{M}$,

$$\Pr[\mathsf{Decrypt}(\mathsf{ct}, C, w) = \mu : \mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, \mu)] = 1.$$

- **Semantic security:** For a security parameter $\lambda \in \mathbb{N}$, a bit $b \in \{0,1\}$, and an adversary $\mathcal{A}$, we define the semantic security game as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs a Boolean relation $C\colon \{0,1\}^h \to \{0,1\}$ and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.
  - If there exists $w \in \{0,1\}^h$ such that $C(w) = 1$, then the challenger outputs 0. Otherwise, the challenger responds with $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, \mu_b)$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is also the output of the experiment.

  The witness encryption scheme is semantically secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the semantic security game.

2.1 Standard Cryptographic Notions

In this section, we recall the formal definitions of some standard cryptographic notions.

Digital signatures and public-key encryption. We begin with the standard notion of a (one-time) digital signature scheme and a public-key encryption scheme.

Definition 2.2 (One-Time Digital Signature)**.** A one-time digital signature scheme Π_{OTS} over a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ is a triple of efficient algorithms $\Pi_{\mathsf{OTS}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm outputs a verification key vk and a signing key sk. We assume that sk and vk implicitly include a description of the security parameter 1^λ.

- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: On input the signing key sk and a message $m \in \mathcal{M}_\lambda$, the signing algorithm outputs a signature σ.

- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to b$: On input the verification key vk, a message m, and a signature σ, the verification algorithm outputs a bit $b \in \{0,1\}$.

We require Π_{OTS} satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$ and all $m \in \mathcal{M}_\lambda$, we have

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1 : \begin{array}{c} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{array}\right].$$

- **One-time strong unforgeability:** For a security parameter λ, an adversary $\mathcal{A}$, we define the one-time strong unforgeability game as follows:
  - The challenger begins by sampling $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and gives vk to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ can make a signing query on a message $m \in \mathcal{M}_\lambda$. The challenger responds with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m^*)$.

  - Algorithm $\mathcal{A}$ outputs a forgery (m^*, σ^*).
  - The challenger outputs $b = 1$ if $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$ and in addition, if $\mathcal{A}$ made a signing query on the message m to receive a signature σ, then $(m^*, \sigma^*) \neq (m, \sigma)$. Otherwise, the challenger outputs $b = 0$.

  We say Π_{OTS} satisfies one-time strong unforgeability if for all existing adversaries $\mathcal{A}$, there exists a negligible function such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the above security game.

Definition 2.3 (Public-Key Encryption)**.** A public-key encryption scheme Π_{PKE} is a triple of efficient algorithms $\Pi_{\mathsf{PKE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: On input a security parameter λ, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{Encrypt}(\mathsf{pk}, m) \to \mathsf{ct}$: On input a public key pk and a message $m \in \{0, 1\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: On input a secret key sk and a ciphertext ct, the decryption algorithm outputs a message $m \in \{0, 1\}$.

We require that Π_{PKE} satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$ and all messages $m \in \{0, 1\}$,
$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m) \end{array}\right] = 1.$$
- **CPA-security:** For a security parameter λ, an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$, we define the CPA-security game as follows:[6]
  - At the beginning of the game, the challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and gives $(1^\lambda, \mathsf{pk})$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ can now make adaptive queries on pairs of messages $m_0, m_1 \in \{0, 1\}$. On each query, the challenger responds with $\mathsf{ct}_b \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b)$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  We say that Π_{PKE} is CPA-secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the CPA-security game.

Leveled homomorphic encryption. A leveled homomorphic encryption [Gen09] enables a bounded number of homomorphic operations on encrypted inputs. We recall the formal definition below:

Definition 2.4 (Leveled Homomorphic Encryption)**.** A public-key (leveled) homomorphic encryption scheme is a tuple of efficient algorithms $\Pi_{\mathsf{LHE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Eval}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, 1^d) \to (\mathsf{pk}, \mathsf{sk})$: On input a security parameter λ and a depth bound d, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{Encrypt}(\mathsf{pk}, m) \to \mathsf{ct}$: On input a public key pk and a message $m \in \{0, 1\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: On input a secret key sk and a ciphertext ct, the decryption algorithm outputs a message $m \in \{0, 1\}$.
- $\mathsf{Eval}(\mathsf{pk}, C, \{\mathsf{ct}_i\}_{i \in [\ell]}) \to \mathsf{ct}'$: On input a Boolean circuit $C \colon \{0, 1\}^\ell \to \{0, 1\}$ and a collection of ℓ ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$, the evaluation algorithm outputs a new ciphertext ct'. This algorithm is deterministic.

[6]Note that we can equivalently define the simpler game where the adversary makes a *single* encryption query, which implies the multi-query version via a standard hybrid argument. However, the multi-query definition is useful in our security proofs.

We require that Π_{LHE} satisfy the following properties:

- **Correctness:** For all $\lambda, d \in \mathbb{N}$, all Boolean circuits $C \colon \{0,1\}^\ell \to \{0,1\}$ of depth at most d, and all inputs $x \in \{0,1\}^\ell$,
$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}') = C(x) : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \forall i \in [\ell] : \mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x_i) \\ \mathsf{ct}' = \mathsf{Eval}(\mathsf{pk}, C, \{\mathsf{ct}_i\}_{i \in [\ell]}) \end{array}\right] = 1.$$

- **Compactness:** There exists a universal polynomial p such that for all $\lambda, d \in \mathbb{N}$, all Boolean circuits $C \colon \{0,1\}^\ell \to \{0,1\}$ with depth at most d, all $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{KeyGen}(1^\lambda, 1^d)$, all inputs $x_1, \ldots, x_\ell \in \{0,1\}$, and all ciphertexts ct_i in the support of $\mathsf{Encrypt}(\mathsf{pk}, x_i)$ for each $i \in [\ell]$, it holds that
$$|\mathsf{pk}| \le p(\lambda, d) \quad \text{and} \quad |\mathsf{ct}'| \le p(\lambda, d),$$
where $\mathsf{ct}' = \mathsf{Eval}(\mathsf{pk}, C, \{\mathsf{ct}_i\}_{i \in [\ell]})$.

- **CPA-security:** For a security parameter λ, an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the CPA-security game as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the depth bound 1^d. The challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^d, 1^s)$ and replies to $\mathcal{A}$ with pk.
  - Algorithm $\mathcal{A}$ can now make adaptive queries on pairs of messages $m_0, m_1 \in \{0,1\}$. On each query, the challenger responds with $\mathsf{ct}_b \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b)$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{LHE} is CPA-secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the CPA-security game.

Remark 2.5 (Encrypting Longer Messages)**.** We extend the encryption algorithm Encrypt to support arbitrary-length messages $m \in \{0,1\}^\ell$ by encrypting bit-by-bit (i.e., for a message $m = m_1 m_2 \cdots m_\ell$, $\mathsf{Encrypt}(\mathsf{pk}, m)$ outputs $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ where $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_i)$). Similarly, when considering CPA-security for longer messages, we allow the adversary to submit arbitrary-length messages m_0, m_1 to the challenger, with the restriction that $|m_0| = |m_1|$.

Somewhere statistically binding hash functions. A somewhere statistically binding hash function [HW15] is a hash function where the digest of an input $x \in \{0,1\}^\ell$ *statistically* binds to the value of x_i at some index $i \in [\ell]$. Moreover, the description of the hash function (i.e., the hash key) computationally hides the index i. In the following definition, we describe a variant that statistically binds to a set of indices (and where the size of the digest scales linearly with the size of the set). We give the formal syntax below:

Definition 2.6 (Somewhere Statistically Binding Hash Function [HW15])**.** A somewhere statistically binding hash function Π_{SSB} is a triple of efficient algorithms $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^{k_{\max}}, n_{\max}, S) \to \mathsf{hk}$: On input a security parameter λ, a block length ℓ_{blk}, a bound on the size of the binding set $k_{\max}$, a bound on the number of blocks $n_{\max}$, and a set $S \subseteq [n_{\max}]$ of size at most $k_{\max}$, the setup algorithm outputs a hash key hk. We assume that hk (implicitly) contains a description of $(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^{k_{\max}}, n_{\max})$.

- $\mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_n)) \to (\mathsf{h}, \pi_1, \ldots, \pi_n)$: On input a hash key hk and a tuple of inputs $x_1, \ldots, x_n \in \{0,1\}^{\ell_{\mathsf{blk}}}$ where $n \le n_{\max}$, the hashing algorithm outputs a hash h together with openings $\pi_1, \ldots, \pi_n$.

- $\mathsf{Verify}(\mathsf{hk}, \mathsf{h}, i, x_i, \pi_i) \to b$: On input a hash key hk, a hash h, an index $i \in [n_{\max}]$, an input $x_i \in \{0,1\}^{\ell_{\mathsf{blk}}}$, and a proof π_i, the verification algorithm outputs a bit $b \in \{0,1\}$.

We require that Π_{SSB} satisfy the following properties:

- **Correctness:** For all $\lambda, \ell_{\text{blk}}, k_{\max}, n_{\max} \in \mathbb{N}$, all input lengths $n \in [n_{\max}]$, all inputs $x_1, \ldots, x_n \in \{0,1\}^{\ell_{\text{blk}}}$, all indices $i \in [n]$, and all sets $S \subseteq [n_{\max}]$ of size at most $k_{\max}$, we have that

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, \mathsf{h}, i, x_i, \pi_i) = 1 : \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\text{blk}}}, 1^{k_{\max}}, n_{\max}, S) \\ (\mathsf{h}, \pi_1, \ldots, \pi_n) \leftarrow \mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_n)) \end{array}\right] = 1.$$

- **Set hiding:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the set-hiding experiment as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the input length $1^{\ell_{\text{blk}}}$, the bound $1^{k_{\max}}$, the number of blocks $n_{\max}$, and a set $S \subseteq [n_{\max}]$ of size at most $k_{\max}$.
  - If $b = 0$, the challenger computes $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\text{blk}}}, 1^{k_{\max}}, n_{\max}, \varnothing)$ and if $b = 1$, it computes $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\text{blk}}}, 1^{k_{\max}}, n_{\max}, S)$. The challenger gives hk to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{SSB} satisfies set hiding if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the set hiding game.

- **Somewhere statistically binding:** We say a hash key hk is statistically binding for a set S if there does not exist values $(\mathsf{h}, i, x_i, x_i', \pi_i, \pi_i')$ where

$$i \in S \quad \text{and} \quad x_i \neq x_i' \quad \text{and} \quad \mathsf{Verify}(\mathsf{hk}, \mathsf{h}, i, x_i, \pi_i) = 1 = \mathsf{Verify}(\mathsf{hk}, \mathsf{h}, i, x_i', \pi_i').$$

  We say that Π_{SSB} is somewhere statistically binding if for all polynomials $\ell_{\text{blk}} = \ell_{\text{blk}}(\lambda)$, $k_{\max} = k_{\max}(\lambda)$, and $n_{\max} = n_{\max}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all sets $S \subseteq [n_{\max}]$ of size at most $k_{\max}$,

$$\Pr\left[\mathsf{hk} \text{ is statistically binding for } S : \ \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\text{blk}}}, 1^{k_{\max}}, n_{\max}, S)\ \right] \geq 1 - \mathsf{negl}(\lambda).$$

- **Succinctness:** There exists a universal polynomial p such that for all $\lambda, \ell_{\text{blk}}, k_{\max}, n_{\max} \in \mathbb{N}$, all sets $S \subseteq [n_{\max}]$ of size at most $k_{\max}$, all inputs $x_1, \ldots, x_n \in \{0,1\}^{\ell_{\text{blk}}}$ where $n \leq n_{\max}$, all hash keys hk in the support of $\mathsf{Setup}(1^\lambda, 1^{\ell_{\text{blk}}}, 1^{k_{\max}}, n_{\max}, S)$, and all $(\mathsf{h}, \pi_1, \ldots, \pi_n)$ in the support of $\mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_n))$, it holds that $|\mathsf{hk}|, |\mathsf{h}|, |\pi_i| \leq p(\lambda, \ell_{\text{blk}}, k_{\max}, \log n_{\max})$ for all $i \in [n]$.

Function-binding hash functions. Function-binding hash functions [FWW23] generalize somewhere-statistically binding hash functions by statistically binding to a (computationally-hidden) *function* of the input. In this work, we specialize our syntax to disjunctions of block functions, which is the primary family of function-binding hash functions considered in [FWW23] and which suffice for our applications. Specifically, we consider hash functions that bind to a disjunction of a function g:

$$f(x_1, \ldots, x_n) := \bigvee_{i \in [n]} g(x_i).$$

Like somewhere statistically binding hash functions, a user can compute a hash of any input $(x_1, \ldots, x_n)$ and produce a succinct local opening of x_i with respect to the hash value. The function binding property (specifically, the statistical disjunction binding property) then asserts that if one computes a hash of $(x_1^*, \ldots, x_n^*)$ where $f(x_1^*, \ldots, x_n^*) = 0$, then there does *not* exist an opening to any $\hat{x}_{i^*}$ where $g(\hat{x}_{i^*}) = 1$ for all i^*. This is because for all values of $\hat{x}_i^*$,

$$f(x_1, \ldots x_{i^*-1}, \hat{x}_{i^*}, x_{i^*+1}, \ldots, x_n) = g(\hat{x}_{i^*}) \vee \bigvee_{i \neq i^*} g(x_i) = 1 \neq f(x_1^*, \ldots, x_n^*).$$

We now give the formal definition adapted from [FWW23].

Definition 2.7 (Function-Binding Hash Function for Disjunction of Block Functions [FWW23]). A function-binding hash function Π_{FBH} for disjunctions of block functions is a triple of efficient algorithms $\Pi_{\mathsf{FBH}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^{d_{\max}}, 1^{s_{\max}}, n_{\max}, C) \to \mathsf{hk}$: On input a security parameter λ, a block length ℓ_{blk}, a depth bound $d_{\max}$, a size bound $s_{\max}$, a bound on the number of blocks $n_{\max}$, and a Boolean circuit $C\colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ (or a special symbol $C = \bot$), the setup algorithm outputs a hash key hk.
- $\mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_n)) \to (\mathsf{h}, \pi_1, \ldots, \pi_n)$: On input a hash key hk and a collection of inputs $x_1, \ldots, x_n \in \{0,1\}^{\ell_{\mathsf{blk}}}$ where $n \le n_{\max}$, the hashing algorithm outputs a hash h together with openings $\pi_1, \ldots, \pi_n$.
- $\mathsf{Verify}(\mathsf{hk}, \mathsf{h}, i, x_i, \pi_i) \to b$: On input a hash key hk, a hash h, an index $i \in [n_{\max}]$, an input $x_i \in \{0,1\}^{\ell_{\mathsf{blk}}}$, and a proof π_i, the verification algorithm outputs a bit $b \in \{0,1\}$.

We require that Π_{FBH} satisfy the following properties:

- **Correctness:** For all $\lambda, \ell_{\mathsf{blk}}, d_{\max}, s_{\max}, n_{\max} \in \mathbb{N}$, all input lengths $n \in [n_{\max}]$, all inputs $x_1, \ldots, x_n \in \{0,1\}^{\ell_{\mathsf{blk}}}$, all indices $i \in [n]$, and all Boolean circuits $C\colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ of depth at most $d_{\max}$ and size at most $s_{\max}$ (or alternatively, $C = \bot$), we have that
$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, \mathsf{h}, i, x_i, \pi_i) = 1 : \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^{d_{\max}}, 1^{s_{\max}}, n_{\max}, C) \\ (\mathsf{h}, \pi_1, \ldots, \pi_n) \leftarrow \mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_n)) \end{array}\right]$$
- **Function hiding:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the function-hiding experiment as follows:
  - On input the security parameter, algorithm $\mathcal{A}$ outputs the input length $1^{\ell_{\mathsf{blk}}}$, the bounds $1^{d_{\max}}$, $1^{s_{\max}}$, and $n_{\max}$, together with a Boolean circuit $C\colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ of depth at most $d_{\max}$ and size at most $s_{\max}$.
  - If $b = 0$, the challenger computes $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^{d_{\max}}, 1^{s_{\max}}, n_{\max}, \bot)$. If $b = 1$, the challenger computes $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^{d_{\max}}, 1^{s_{\max}}, n_{\max}, C)$. The challenger gives hk to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{FBH} satisfies function hiding if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the function hiding game.
- **Statistically disjunction binding:** Fix parameters $\ell_{\mathsf{blk}}, d_{\max}, s_{\max}, n_{\max}$. We say a hash key hk is statistically-disjunction-binding for a block function $C\colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ if for all inputs $x_1, \ldots, x_n \in \{0,1\}^{\ell_{\mathsf{blk}}}$ where $n \le n_{\max}$ and $C(x_i) = 0$ for all $i \in [n]$, and setting $(\mathsf{h}, \pi_1, \ldots, \pi_n) = \mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_n))$, there does not exist an opening (i, x_i, π_i) where
$$C(x_i) = 1 \quad \text{and} \quad \mathsf{Verify}(\mathsf{hk}, \mathsf{h}, i, x_i, \pi_i).$$
  We say that Π_{FBH} is statistically disjunction binding if for all polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, $k_{\max} = k_{\max}(\lambda)$, $s_{\max} = s_{\max}(\lambda)$, and $n_{\max} = n_{\max}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all Boolean circuits $C\colon \{0,1\}^{\ell_{\mathsf{blk}}}$ of depth at most $d_{\max}$ and size at most $s_{\max}$,
$$\Pr[\mathsf{hk} \text{ is statistically-disjunction-binding for } C : \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^{d_{\max}}, 1^{s_{\max}}, n_{\max}, C)] \ge 1 - \mathsf{negl}(\lambda).$$
- **Succinctness:** There exists universal polynomials p_1, p_2 such that for all $\lambda, \ell_{\mathsf{blk}}, k_{\max}, d_{\max}, n_{\max} \in \mathbb{N}$, all Boolean circuits $C\colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ of depth at most $d_{\max}$ and size at most $s_{\max}$ (or alternatively, $C = \bot$), all inputs $x_1, \ldots, x_n \in \{0,1\}^{\ell_{\mathsf{blk}}}$ where $n \le n_{\max}$, all hash keys hk in the support of $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, 1^{d_{\max}}, 1^{s_{\max}}, n_{\max}, C)$, and all $(\mathsf{h}, \pi_1, \ldots, \pi_n)$ in the support of $\mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_n))$, it holds that
  - $|\mathsf{hk}| \le p_1(\lambda, s_{\max}, \log n_{\max})$.
  - $|\mathsf{h}|, |\pi_i| \le p_2(\lambda, d_{\max}, \log s_{\max}, \log n_{\max})$.

Batch arguments for NP. A non-interactive batch argument (BARG) for NP [BHK17, CJJ21a, CJJ21b] allows a prover to convince a verifier that a batch of k NP statements $x_1, \ldots, x_k$ are true with a proof whose size scales sublinearly with k. In this work, we will consider the specific setting of an index BARG [CJJ21b], which corresponds to the special case of index languages (i.e., the batch language where the statements are simply the indices $1, 2, \dots, k$). Moreover, we consider the stronger notion of somewhere *statistical* soundness where there does not *exist* a valid proof π with respect to a CRS that is binding on index i when statement i is *false*. The [WW22] construction satisfies this stronger notion of somewhere statistical soundness; other BARG constructions [CJJ21b, CGJ+23] have been shown to satisfy a weaker computational version of this property. We review the formal definition below:

Definition 2.8 (Non-Interactive Batch Argument for Index Languages)**.** An non-interactive batch argument for index languages Π_{BARG} is a triple of efficient algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{k_{\max}}, 1^{s_{\max}}) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, a bound on the number of instances $k_{\max} \in \mathbb{N}$, and a bound on the circuit size $s_{\max} \in \mathbb{N}$, the setup algorithm outputs a common reference string crs. We assume that crs(implicitly) contain a description of $(1^\lambda, k_{\max}, s_{\max})$.
- $\mathsf{Prove}(\mathsf{crs}, C, (w_1, \dots, w_k)) \to \pi$: On input the common reference string crs, a Boolean circuit C of size at most $s_{\max}$, together with $k \le k_{\max}$ witnesses $w_1, \dots, w_k$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, C, k, \pi) \to b$: On input the common reference string crs, a Boolean circuit C of size at most $s_{\max}$, an integer $k \le k_{\max}$, and a proof π, the verification algorithm outputs a bit $b \in \{0, 1\}$.

We require that Π_{BARG} satisfy the following properties:

- **Completeness:** For all $\lambda, k_{\max}, s_{\max} \in \mathbb{N}$, all Boolean circuits C of size at most $s_{\max}$, all positive integers $k \le k_{\max}$, all witnesses $w_1, \dots, w_k$ where $C(i, w_i) = 1$ for all $i \in [k]$, it holds that
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, C, k, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{k_{\max}}, 1^{s_{\max}}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (w_1, \dots, w_k)) \end{array}\right] = 1.$$
- **Somewhere statistical soundness:** There exists an efficient algorithm Setup^* with the following syntax:
  - $\mathsf{Setup}^*(1^\lambda, 1^{k_{\max}}, 1^{s_{\max}}, i) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, a bound on the number of instances $k_{\max} \in \mathbb{N}$, a bound on the circuit size $s_{\max} \in \mathbb{N}$, and an index $i \in [k_{\max}]$, the setup algorithm outputs a common reference string crs. We assume that the CRS (implicitly) contain a description of $(1^\lambda, k_{\max}, s_{\max})$.

  We additionally require the following properties:

  - **Mode indistinguishability:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0, 1\}$, we define the mode indistinguishability game as follows:
    - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs $1^{k_{\max}}$ and $1^{s_{\max}}$ and an index $i \in [k_{\max}]$.
    - If $b = 0$, the challenger replies with $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{k_{\max}}, 1^{s_{\max}})$. If $b = 1$, the challenger replies with $\mathsf{crs} \leftarrow \mathsf{Setup}^*(1^\lambda, 1^{k_{\max}}, 1^{s_{\max}}, i)$.
    - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ which is the output of the experiment.

    We say that Π_{BARG} satisfies mode indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the mode indistinguishability game.
  - **Somewhere statistical soundness:** We say that Π_{BARG} satisfies somewhere statistical soundness if for all polynomials $k_{\max} = k_{\max}(\lambda)$ and $s_{\max} = s_{\max}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $i \in [k_{\max}]$, all Boolean circuits of size at most $s_{\max}$ where $C(i, w) = 0$ for all $w \in \{0, 1\}^*$, all $i \le k \le k_{\max}$, and all $\lambda \in \mathbb{N}$,
$$\Pr[\exists \pi : \mathsf{Verify}(\mathsf{crs}, C, k, \pi) = 1 \mid \mathsf{crs} \leftarrow \mathsf{Setup}^*(\mathsf{crs}, 1^{k_{\max}}, 1^{s_{\max}}, i)] = \mathsf{negl}(\lambda).$$
- **Succinctness:** There exists a universal polynomial p such that for all $\lambda, k_{\max}, s_{\max} \in \mathbb{N}$, all Boolean circuits C of size at most $s_{\max}$, all positive integers $k \le k_{\max}$, all witnesses $w_1, \dots, w_k$ where $C(i, w_i) = 1$ for all $i \in [k]$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^{k_{\max}}, 1^{s_{\max}})$ and all proofs π in the support of $\mathsf{Prove}(\mathsf{crs}, C, (w_1, \dots, w_k))$, we have that $|\mathsf{crs}| \le p(\lambda, \log s_{\max}, \log k_{\max})$, and $|\pi| \le p(\lambda, |C|, \log k)$.

Indistinguishability obfuscation. We recall the notion of an indistinguishability obfuscation scheme [BGI+01]:

Definition 2.9 (Indistinguishability Obfuscation [BGI+01])**.** An indistinguishability obfuscator for Boolean circuits is an efficient algorithm $i\mathcal{O}(\cdot,\cdot,\cdot)$ with the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, circuit size parameters $s \in \mathbb{N}$, all Boolean circuits C of size at most s, and all inputs x,
$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(1^\lambda, 1^s, C)] = 1.$$

- **Security:** For a bit $b \in \{0,1\}$ and a security parameter λ, we define the program indistinguishability game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input the security parameter 1^λ, the adversary outputs a size parameter 1^s and two Boolean circuits C_0, C_1 of size at most s.
  - If there exists an input x such that $C_0(x) \neq C_1(x)$, then the challenger halts with output $\perp$. Otherwise, the challenger replies with $i\mathcal{O}(1^\lambda, 1^s, C_b)$.
  - The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that $i\mathcal{O}$ is secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that
$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \mathsf{negl}(\lambda)$$
  in the above program indistinguishability game.

Puncturable PRFs and injective PRGs. Next, we recall the notion of a puncturable PRF [BW13, KPTZ13, BGI14] and the notion of an injective PRG, which will be useful in combination with $i\mathcal{O}$ to obtain our succinct unique witness map for batch languages in Section 5.

Definition 2.10 (Puncturable PRF [BW13, KPTZ13, BGI14])**.** A puncturable pseudorandom function consists of a tuple of efficient algorithms $\Pi_{\mathsf{PPRF}} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Puncture})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathrm{in}}}, 1^{\ell_{\mathrm{out}}}) \to k$: On input the security parameter λ, an input length ℓ_{in}, and an output length ℓ_{out}, the key-generation algorithm outputs a key k. We assume that the key k contains an implicit description of ℓ_{in} and ℓ_{out}.

- $\mathsf{Puncture}(k, x^*) \to k^{(x^*)}$: On input a key k and a point $x^* \in \{0,1\}^{\ell_{\mathrm{in}}}$, the puncture algorithm outputs a punctured key $k^{(x^*)}$. We assume the punctured key contains an implicit description of ℓ_{in} and ℓ_{out}.

- $\mathsf{Eval}(k, x) \to y$: On input a key k and an input $x \in \{0,1\}^{\ell_{\mathrm{in}}}$, the evaluation algorithm outputs a value $y \in \{0,1\}^{\ell_{\mathrm{out}}}$:

In addition, Π_{PPRF} should satisfy the following properties:

- **Functionality-preserving:** For all $\lambda, \ell_{\mathrm{in}}, \ell_{\mathrm{out}} \in \mathbb{N}$, every input $x \in \{0,1\}^{\ell_{\mathrm{in}}}$, and every $x \in \{0,1\}^{\ell_{\mathrm{in}}} \setminus \{x^*\}$,
$$\Pr\left[\mathsf{Eval}(k, x) = \mathsf{Eval}(k^{(x^*)}, x) : \begin{array}{c} k \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathrm{in}}}, 1^{\ell_{\mathrm{out}}}) \\ k^{(x^*)} \leftarrow \mathsf{Puncture}(k, x^*) \end{array}\right] = 1.$$

- **Punctured pseudorandomness:** For a bit $b \in \{0,1\}$ and a security parameter λ, we define the (selective) punctured pseudorandomness game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the input length $1^{\ell_{\mathrm{in}}}$, the output length $1^{\ell_{\mathrm{out}}}$, and commits to a challenge point $x^* \in \{0,1\}^{\ell_{\mathrm{in}}}$.
  - The challenger samples $k \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathrm{in}}}, 1^{\ell_{\mathrm{out}}})$ and gives $k^{(x^*)} \leftarrow \mathsf{Puncture}(k, x^*)$ to $\mathcal{A}$.
  - If $b = 0$, the challenger gives $y^* = \mathsf{Eval}(k, x^*)$ to $\mathcal{A}$. If $b = 1$, then it gives $y^* \xleftarrow{\mathrm{R}} \{0,1\}^{\ell_{\mathrm{out}}}$ to $\mathcal{A}$.

- At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say that Π_{PPRF} satisfies punctured pseudorandomness if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \le \mathsf{negl}(\lambda)$$

in the punctured pseudorandomness security game.

Definition 2.11 (Injective PRG)**.** Let $\ell = \ell(\lambda)$ be an input-length parameter and $m = m(\lambda)$ be an output length parameter. An injective pseudorandom generator (PRG) is an efficiently-computable injective function $\mathsf{G} \colon \{0,1\}^{\ell} \to \{0,1\}^{m}$ where for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{A}(1^\lambda, \mathsf{G}(s)) = 1 : s \xleftarrow{\mathrm{R}} \{0,1\}^{\ell}] - \Pr[\mathcal{A}(1^\lambda, t) = 1 : t \xleftarrow{\mathrm{R}} \{0,1\}^{m}]| = \mathsf{negl}(\lambda).$$

3 Succinct Witness Encryption for Batch Languages

We begin by introducing the notion of succinct witness encryption for batch languages. This is the main cryptographic notion we consider in this work, and in Section 6, we show how it can be used to realize applications to computational secret sharing and monotone-policy encryption.

Definition 3.1 (Succinct Witness Encryption for Batch Languages)**.** Let $\mathcal{M}$ be a message space and $\mathcal{P}$ be a family of policies. A succinct witness encryption scheme Π_{WE} for batch languages with message space $\mathcal{M}$ and policy family $\mathcal{P}$ is a pair of efficient algorithms $\Pi_{\mathsf{WE}} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu) \to \mathsf{ct}$: On input the security parameter $\lambda \in \mathbb{N}$, a Boolean relation $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a Boolean policy $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and a message μ, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{ct}, C, P, (x_1, \ldots, x_K), (w_1, \ldots, w_K)) \to \mu$: On input a ciphertext ct, a Boolean relation $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a Boolean policy $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, statements $(x_1, \ldots, x_K) \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_K \in \{0,1\}^h$, the decryption algorithm outputs a message μ.

Moreover, we require that Π_{WE} satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all Boolean relations $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all Boolean policies $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, all tuples of statements $x_1, \ldots, x_K \in \{0,1\}^n$, all witnesses $w_1, \ldots, w_K \in \{0,1\}^h$ where $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$, and all messages $\mu \in \mathcal{M}$,

$$\Pr[\mathsf{Decrypt}(\mathsf{ct}, C, P, (x_1, \ldots, x_K), (w_1, \ldots, w_K)) = \mu : \mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu)] = 1.$$

- **Semantic security:** For a security parameter $\lambda \in \mathbb{N}$, a bit $b \in \{0,1\}$, and an adversary $\mathcal{A}$, we define the semantic security game as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs a Boolean relation $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a policy $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, a tuple of statements $x_1, \ldots, x_K \in \{0,1\}^n$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$.
  - If there exists $w_1, \ldots, w_K \in \{0,1\}^h$ such that $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$, then the challenger outputs 0. Otherwise, the challenger responds with the ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu)$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

The succinct witness encryption scheme for batch languages is semantically secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$$

in the semantic security game.

- **Succinctness:** There exists a polynomial poly such that for all $\lambda \in \mathbb{N}$, circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, policies $P \in \mathcal{P}$ (on n-bit inputs), instances $x_1, \ldots, x_K \in \{0,1\}^n$, and messages $\mu \in \mathcal{M}$, the size of the ciphertext ct output by $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu)$ satisfies $|\mathsf{ct}| \le o(|P|) \cdot \mathsf{poly}(\lambda, |C|, \log K)$.

Local decryption. In Definition 3.1, the decryption algorithm requires knowledge of *all* of the statements associated with the ciphertext. When considering applications of succinct witness encryption to computational secret sharing and monotone policy encryption, it will be important to consider a decryption algorithm that only requires knowledge of a *subset* of statements that satisfy the policy (as opposed to all of the statements). Similar to the setting of batch arguments [CJJ21b] and locally verifiable signatures, we can support this property by decomposing the decryption algorithm into a preprocessing algorithm which takes as input the policy P together with *all* of the statements and outputs a short "hint" associated with each statement. The preprocessing algorithm only depends on the statement and not the witnesses. Then, there is a local decryption algorithm that takes as input the ciphertext together with a subset of statements and their associated hints and witnesses and outputs the message. Notably, the local decryption algorithm only requires knowledge of the statements and hints that satisfy the policy. We define this property formally below:

Definition 3.2 (Local Decryption)**.** A succinct witness encryption scheme $\Pi_{\mathsf{WE}} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ with message space $\mathcal{M}$ and policy family $\mathcal{P}$ supports local decryption if there exist a pair of efficient algorithms $(\mathsf{Preprocess}, \mathsf{DecryptLocal})$ with the following syntax:

- $\mathsf{Preprocess}(\mathsf{ct}, C, P, (x_1, \ldots, x_K)) \to (\mathsf{ht}_1, \ldots, \mathsf{ht}_K)$: On input a ciphertext ct, a Boolean relation $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a Boolean policy $P \in \mathcal{P}$, and the statements $x_1, \ldots, x_K \in \{0,1\}^n$, the preprocessing algorithm outputs a tuple of hints $\mathsf{ht}_1, \ldots, \mathsf{ht}_K$. This algorithm is deterministic.
- $\mathsf{DecryptLocal}(\mathsf{ct}, C, P, \{(i, \mathsf{ht}_i, w_i)\}_{i \in S}) \to \mu$: On input a ciphertext ct, a Boolean relation $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a Boolean policy $P \in \mathcal{P}$, and a collection of hints and witnesses (ht_i, w_i) for $i \in S$, the local decryption algorithm outputs a message μ. This algorithm is also deterministic.

We require $(\mathsf{Preprocess}, \mathsf{DecryptLocal})$ satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all Boolean relations $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all Boolean policies $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, all inputs $\beta_1, \ldots, \beta_K \in \{0,1\}$ where $P(\beta_1, \ldots, \beta_K) = 1$, all statements $x_1, \ldots, x_K \in \{0,1\}^n$, all witnesses $w_1, \ldots, w_K \in \{0,1\}^h$ where $C(x_i, w_i) = 1$ for all i where $\beta_1 = 1$, and for all messages $\mu \in \mathcal{M}$,
$$\Pr\left[\mathsf{DecryptLocal}(\mathsf{ct}, C, P, \{(i, \mathsf{ht}_i, w_i)\}_{i \in [K] : \beta_i = 1}) = \mu\right] = 1,$$
where $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu)$ and $(\mathsf{ht}_1, \ldots, \mathsf{ht}_K) = \mathsf{Preprocess}(\mathsf{ct}, C, P, (x_1, \ldots, x_K))$.
- **Succinct hints:** There exists a polynomial poly such that for all $\lambda \in \mathbb{N}$, circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, policies $P \in \mathcal{P}$ (on n-bit inputs), instances $x_1, \ldots, x_K \in \{0,1\}^n$, messages $\mu \in \mathcal{M}$, and all ciphertexts ct in the support of $\mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu)$, the hints $(\mathsf{ht}_1, \ldots, \mathsf{ht}_K) = \mathsf{Preprocess}(\mathsf{ct}, C, P, (x_1, \ldots, x_K))$ satisfy
$$\forall i \in [K] : |\mathsf{ht}_i| \le o(|P|) \cdot \mathsf{poly}(\lambda, |C|, \log K).$$

4 Succinct Witness Encryption for CNFs and DNFs

In this section, we show how to construct succinct witness encryption with succinct ciphertexts that support CNF and DNF policies from witness encryption (in conjunction with either somewhere statistically sound batch arguments for NP [WW22] or function-binding hash functions [FWW23]).

4.1 Succinct Witness Encryption for CNF Policies

We start by constructing a succinct witness encryption scheme that supports CNF policies. The size of the ciphertext scales with the maximum number of variables that can appear in a single clause but polylogarithmically in the total number of clauses. For the particular setting where each clause contains a constant number of variables, the size of the ciphertext (and public parameters) scale polylogarithmically with the description length of the CNF.

Construction 4.1 (Succinct Witness Encryption for CNF Policies)**.** Let λ be a security parameter, $\mathcal{M}$ be a message space, and $\mathcal{P}$ be the set of Boolean formulas in conjunctive normal form. Our construction relies on the following:

- Let $\Pi_{\mathsf{SSB}} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Verify})$ be a somewhere statistically binding hash function.
- Let $\Pi_{\mathsf{BARG}} = (\mathsf{BARG.Setup}, \mathsf{BARG.Prove}, \mathsf{BARG.Verify})$ be a somewhere statistically sound index BARG.
- Let $\Pi_{\mathsf{WE}} = (\mathsf{WE.Encrypt}, \mathsf{WE.Decrypt})$ be a witness encryption scheme with message space $\mathcal{M}$.

We construct a succinct witness encryption scheme with message space $\mathcal{M}$ and policy space $\mathcal{P}$ as follows:

- $\mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu)$: On input the security parameter λ, the Boolean relation $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, the Boolean policy $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and the message $\mu \in \mathcal{M}$, the encryption algorithm proceeds as follows:
  - Let c be the number of clauses in P and let $t \le K$ be the maximum number of variables that appears in any clause of P.
  - Sample $\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^t, K, \varnothing)$. Let $\ell_{\mathsf{cl}} = \ell_{\mathsf{cl}}(\lambda)$ be a bound on the description of a single clause. Sample $\mathsf{hk}_{\mathsf{cl}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{cl}}}, 1^1, c, \varnothing)$.
  - Compute a hash of the instances
  $$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)).$$
  - For $i \in [c]$, let $S_i \subseteq [K]$ denote the variables that appear in clause i of P (under a canonical ordering of the variables). Compute a hash of the clauses $(\mathsf{h}_{\mathsf{cl}}, \pi_{\mathsf{cl},1}, \ldots, \pi_{\mathsf{cl},c}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{cl}}, (S_1, \ldots, S_c))$.
  - Define the index relation $\mathcal{R}_{\mathsf{ClauseSAT}}$:

    Fixed values: hash keys $\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}$, a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and hashes $\mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}$
    Statement: index $i \in \mathbb{N}$
    Witness: a set S, an opening π_{cl}, an index j, an NP statement x, an opening π_{inst}, and an NP witness w

    On input a statement $i \in \mathbb{N}$ and a tuple $(S, \pi_{\mathsf{cl}}, j, x, \pi_{\mathsf{inst}}, w)$, output 1 if the following conditions hold:

    * $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{cl}}, \mathsf{h}_{\mathsf{cl}}, i, S, \pi_{\mathsf{cl}}) = 1$.
    * $j \in S$ and $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, j, x, \pi_{\mathsf{inst}}) = 1$ and $C(x, w) = 1$.

    Figure 1: The NP relation $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$

  - Let s be the size of the Boolean circuit $C_{\mathsf{ClauseSAT}}$ that computes $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$ from Fig. 1. Sample $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^c, 1^s)$.
  - Let $C_{\mathsf{ValidBARG}}$ be the Boolean circuit that takes as input a BARG proof π and outputs
  $$\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi).$$

  In particular, the values of $\mathsf{crs}_{\mathsf{BARG}}$, $C_{\mathsf{ClauseSAT}}$, and c are hard-wired in the circuit $C_{\mathsf{ValidBARG}}$. Compute the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidBARG}}, \mu)$.

Output the ciphertext $\mathsf{ct} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{ct}_{\mathsf{WE}})$.

- $\mathsf{Decrypt}(\mathsf{ct}, C, P, (x_1, \ldots, x_K), (w_1, \ldots, w_K))$: On input a ciphertext $\mathsf{ct} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}, \mathsf{ct}_{\mathsf{WE}})$, a Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a policy $P \in \mathcal{P}$ where $P\colon \{0,1\}^K \to \{0,1\}$, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_K \in \{0,1\}^h$, the decryption algorithm proceeds as follows:
  - If $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 0$, output $\perp$.
  - Otherwise, compute a hash of the instances $(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$. Let c be the number of clauses that appear in P. For $i \in [c]$, let $S_i \subseteq [K]$ denote the variables that appear in clause i of P (under a canonical ordering of the variables). Compute a hash of the clauses $(\mathsf{h}_{\mathsf{cl}}, \pi_{\mathsf{cl},1}, \ldots, \pi_{\mathsf{cl},c}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{cl}}, (S_1, \ldots, S_c))$.
  - Since $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$ and P is a CNF, every clause of P must contain a satisfied literal. In other words, for every $i \in [c]$, there exists an index $j_i \in S_i$ where $C(x_{j_i}, w_{j_i}) = 1$. Let j_i be the smallest such index and define $w_{\mathsf{BARG},i} = (S_i, \pi_{\mathsf{cl},i}, j_i, x_{j_i}, \pi_{\mathsf{inst},j_i}, w_{j_i})$.
  - Let $C_{\mathsf{ClauseSAT}}$ be the Boolean circuit that computes the NP relation $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$ from Fig. 1. Construct a proof $\pi_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Prove}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, (w_{\mathsf{BARG},1}, \ldots, w_{\mathsf{BARG},c}))$.
  - Let $C_{\mathsf{ValidBARG}}$ be the circuit that takes as input a proof π and outputs $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi)$. Output $\mathsf{WE.Decrypt}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidBARG}}, \pi_{\mathsf{BARG}})$.

Theorem 4.2 (Correctness). *If Π_{SSB} and Π_{WE} are correct and Π_{BARG} is complete, then Construction 4.1 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, any Boolean policy $P \in \mathcal{P}$ where $P\colon \{0,1\}^n \to \{0,1\}$, any collection of statements $x_1, \ldots, x_K \in \{0,1\}^n$ and witnesses $w_1, \ldots, w_K \in \{0,1\}^h$ where $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$, and any message $\mu \in \mathcal{M}$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, P, (x_1 \ldots, x_K), \mu)$ and consider $\mathsf{Decrypt}(\mathsf{ct}, C, P, (x_1, \ldots, x_K), (w_1, \ldots, w_K))$:

- By construction, $\mathsf{ct} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{ct}_{\mathsf{WE}})$.
- Let c be the number of clauses in P, and for $i \in [c]$, let $S_i \subseteq [K]$ be the variables that appear in the i^{th} clause of P. Since $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$ and P is a CNF, this means that for every $i \in [c]$, there exists an index $j_i \in S_i$ such that $C(x_{j_i}, w_{j_i}) = 1$. As in Encrypt and Decrypt, let j_i be the smallest such index.
- Next, the ciphertext ct satisfies $\mathsf{ct} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidBARG}}, \mu)$ where $C_{\mathsf{ValidBARG}}$ takes as input a BARG proof π and outputs $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi)$.
- By construction, $\mathsf{h}_{\mathsf{inst}}$ is a hash of the instances $(x_1, \ldots, x_K)$ under $\mathsf{hk}_{\mathsf{inst}}$ and h_{cl} is a hash of the sets $S_1, \ldots, S_c$ under $\mathsf{hk}_{\mathsf{cl}}$. The decryption algorithm first computes
$$(\mathsf{h}'_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},n}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$$
$$(\mathsf{h}'_{\mathsf{cl}}, \pi_{\mathsf{cl},1}, \ldots, \pi_{\mathsf{cl},c}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{cl}}, (S_1, \ldots, S_c)).$$
Since SSB.Hash is deterministic and Encrypt and Decrypt compute $\mathsf{hk}_{\mathsf{inst}}$ and $\mathsf{hk}_{\mathsf{cl}}$ (resp., $\mathsf{hk}'_{\mathsf{inst}}, \mathsf{hk}'_{\mathsf{cl}}$) using identical procedures, this means $\mathsf{hk}'_{\mathsf{inst}} = \mathsf{hk}_{\mathsf{inst}}$ and $\mathsf{hk}'_{\mathsf{cl}} = \mathsf{hk}_{\mathsf{cl}}$. For each $i \in [c]$, consider the witness $w_{\mathsf{BARG},i} = (S_i, \pi_{\mathsf{cl},i}, j_j, x_{j_i}, \pi_{\mathsf{inst},j_i}, w_{j_i})$ constructed by the decryption algorithm. By construction, S_i is the i^{th} clause of P and $j_i \in S_i$. By correctness of Π_{SSB}, it holds that
$$\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{cl}}, \mathsf{h}_{\mathsf{cl}}, i, S_i, \pi_{\mathsf{cl},i}) = 1 = \mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, j_i, x_{j_i}, \pi_{\mathsf{inst},j_i}).$$
Moreover, $j_i \in S_i$ and $C(x_{j_i}, w_{j_i}) = 1$. Thus, for all $i \in [c]$,
$$C_{\mathsf{ClauseSAT}}(i, w_{\mathsf{BARG},i}) = \mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}](i, w_{\mathsf{BARG},i}) = 1.$$
- By completeness of Π_{BARG}, this means that $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi_{\mathsf{BARG}}) = 1$ when $\pi_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Prove}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, (w_{\mathsf{BARG},1}, \ldots, w_{\mathsf{BARG},c}))$.

- Correspondingly, $C_{\mathsf{ValidBARG}}(\pi_{\mathsf{BARG}}) = 1$ so by correctness of Π_{WE}, $\mathsf{WE.Decrypt}(\mathsf{ct}_{\mathsf{WE}}, \pi_{\mathsf{BARG}}) = \mu$.

Correctness holds. □

Theorem 4.3 (Semantic Security). *Suppose Π_{SSB} satisfies correctness, set hiding, and somewhere statistical binding, Π_{BARG} is somewhere extractable, and Π_{WE} satisfies semantic security. Then Construction 4.1 is semantically secure.*

Proof. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an efficient non-uniform adversary for the semantic security game. In particular, on input the security parameter 1^λ, algorithm $\mathcal{A}_0$ outputs a tuple $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1)$ together with some state information $\mathsf{st}_{\mathcal{A}}$ (of polynomial size). Algorithm $\mathcal{A}_1$ takes as input the state $\mathsf{st}_{\mathcal{A}}$ and a ciphertext ct and outputs a bit $b' \in \{0, 1\}$. For each $i \in [K]$, define the bit β_i as follows:

$$\beta_i := \begin{cases} 1 & \exists w_i \in \{0,1\}^h : C(x_i, w_i) = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Our reduction algorithms will take the bits $(\beta_1, \ldots, \beta_K)$ along with $\mathsf{st}_{\mathcal{A}}$ as non-uniform advice. We now define a sequence of hybrid experiments parameterized by a bit $b \in \{0, 1\}$:

- $\mathsf{Hyb}_0^{(b)}$: This is the semantic security game with adversary $\mathcal{A}$ and bit $b \in \{0, 1\}$:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}_0(1^\lambda)$ outputs $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1)$ and $\mathsf{st}_{\mathcal{A}}$. For each $i \in [K]$, define the bits β_i according to Eq. (4.1).
  - If $P(\beta_1, \ldots, \beta_K) = 1$, the challenger outputs 0. Otherwise, the challenger invokes $b' \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, \mathsf{ct})$ where $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu_b)$. Specifically, the challenger constructs ct as follows:
    * Let c be the number of clauses in P and let $t \le K$ be the maximum number of variables that appears in any clause of P.
    * The challenger samples $\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^t, K, \varnothing)$ and $\mathsf{hk}_{\mathsf{cl}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{cl}}}, 1^1, c, \varnothing)$, where $\ell_{\mathsf{cl}} = \ell_{\mathsf{cl}}(\lambda)$ is a bound on the description of a single clause.
    * Let s be the size of the circuit that computes $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$ from Fig. 1. The challenger samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^c, 1^s)$.
    * The challenger computes
    $$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$$
    $$(\mathsf{h}_{\mathsf{cl}}, \pi_{\mathsf{cl},1}, \ldots, \pi_{\mathsf{cl},c}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{cl}}, (S_1, \ldots, S_c)),$$
    where $S_i \subseteq [K]$ denotes the variables that appear in clause i.
    * Finally, the challenger computes the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidBARG}}, \mu_b)$, where $C_{\mathsf{ValidBARG}}$ is the circuit that takes π as input and outputs $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi)$.
    * The challenger sets $\mathsf{ct} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{ct}_{\mathsf{WE}})$.
  - The output of the experiment is the bit $b' \in \{0, 1\}$.
- $\mathsf{Hyb}_1^{(b)}$: Suppose $P(\beta_1, \ldots, \beta_K) = 0$. Then, there exists an index $i^* \in [c]$ such that for all $j \in S_{i^*}$, $\beta_j = 0$. Let i^* be the first such index where this property holds. This experiment is the same as $\mathsf{Hyb}_0^{(b)}$ except the challenger samples $\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^t, K, S_{i^*})$.
- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$ except the challenger samples $\mathsf{hk}_{\mathsf{cl}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{cl}}}, 1^1, c, \{i^*\})$.
- $\mathsf{Hyb}_3^{(b)}$: Same as $\mathsf{Hyb}_2^{(b)}$ except the challenger sample $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}^*(1^\lambda, 1^c, 1^s, i^*)$.

We write $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ to denote the output distribution of an adversary $\mathcal{A}$ in experiment $\mathsf{Hyb}_i^{(b)}$. In our reduction algorithms below, we will construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$. In all cases, algorithm $\mathcal{B}_0$ behaves as follows:

On input the security parameter $\lambda \in \mathbb{N}$:

- Run $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_0(1^\lambda)$.
- For each $i \in [K]$, compute β_i according to Eq. (4.1).
- Output $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, (\beta_1, \ldots, \beta_K))$.

Figure 2: The pre-processing algorithm $\mathcal{B}_0$

We now analyze each adjacent pair of hybrid experiments.

Lemma 4.4. *If Π_{SSB} satisfies set hiding, then for all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ for the set-hiding security game. The behavior of $\mathcal{B}_0$ is shown in Fig. 2. On input the non-uniform advice $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, (\beta_1, \ldots, \beta_K))$, algorithm $\mathcal{B}_1$ works as follows:

1. On input the security parameter 1^λ and the advice string $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, (\beta_1, \ldots, \beta_K))$, algorithm $\mathcal{B}_1$ outputs 0 if $P(\beta_1, \ldots, \beta_K) = 1$.

2. Otherwise, interpret $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and $P \colon \{0,1\}^K \to \{0,1\}$. Let c be the number of clauses in P, $t \leq K$ be the maximum number of variables that appears in any clause of P, and $S_i \subseteq [K]$ be the variables that appear in clause i of P. Let $i^* \in [c]$ be the first index where for all $j \in S_{i^*}$, $\beta_j = 0$.

3. Algorithm $\mathcal{B}_1$ outputs the input length 1^n, the bound 1^t, the number of blocks K, and the set S_{i^*}. The challenger replies with a hash key $\mathsf{hk}_{\mathsf{inst}}$. Algorithm $\mathcal{B}_1$ then samples $\mathsf{hk}_{\mathsf{cl}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{cl}}}, 1^1, c, \varnothing)$.

4. Algorithm $\mathcal{B}_1$ computes
$$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$$
$$(\mathsf{h}_{\mathsf{cl}}, \pi_{\mathsf{cl},1}, \ldots, \pi_{\mathsf{cl},c}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{cl}}, (S_1, \ldots, S_c)).$$
Next, it samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^c, 1^s)$, where s is the size of the circuit $C_{\mathsf{ClauseSAT}}$ that computes $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$ from Fig. 1. Finally, algorithm $\mathcal{B}$ constructs the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidBARG}}, \mu_b)$, where $C_{\mathsf{ValidBARG}}$ is the Boolean circuit that takes as input a BARG proof π and outputs $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi)$.

5. Algorithm $\mathcal{B}$ sets $\mathsf{ct} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{ct}_{\mathsf{WE}})$ and outputs $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{ct})$.

If the challenger samples $\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^t, K, \varnothing)$, then algorithm $\mathcal{B}$ simulates an execution of $\mathsf{Hyb}_0^{(b)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]$. If the challenger samples $\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^t, K, S_{i^*})$, then algorithm $\mathcal{B}$ simulates an execution of $\mathsf{Hyb}_1^{(b)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]$. We conclude that algorithm $\mathcal{B}$ breaks set hiding with the same advantage ε. □

Lemma 4.5. *If Π_{SSB} satisfies set hiding, then for all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Follows by a similar argument as the proof of Lemma 4.4, except the reduction algorithm obtains the hash key $\mathsf{hk}_{\mathsf{cl}}$ from the challenger instead of $\mathsf{hk}_{\mathsf{inst}}$. □

Lemma 4.6. *If Π_{BARG} satisfies somewhere statistical soundness (specifically, mode indistinguishability), then for all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*
$$|\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Suppose $|\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct an efficient (non-uniform) adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ for the mode indistinguishability game. The behavior of $\mathcal{B}_0$ is shown in Fig. 2. On input the non-uniform advice $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, (\beta_1, \ldots, \beta_K))$, algorithm $\mathcal{B}_1$ works as follows:

1. On input the security parameter 1^λ and the advice string $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, (\beta_1, \ldots, \beta_K))$, algorithm $\mathcal{B}_1$ outputs 0 if $P(\beta_1, \ldots, \beta_K) = 1$.

2. Otherwise, interpret $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and $P \colon \{0,1\}^K \to \{0,1\}$. Let c be the number of clauses in P, $t \leq K$ be the maximum number of variables that appears in any clause of P, and $S_i \subseteq [K]$ be the variables that appear in clause i of P. Let $i^* \in [c]$ be the first index where for all $j \in S_{i^*}$, $\beta_j = 0$.

3. Algorithm $\mathcal{B}_1$ samples $\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^t, K, S_{i^*})$ and $\mathsf{hk}_{\mathsf{cl}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{cl}}}, 1^1, c, \{i^*\})$.

4. Algorithm $\mathcal{B}_1$ computes

$$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$$
$$(\mathsf{h}_{\mathsf{cl}}, \pi_{\mathsf{cl},1}, \ldots, \pi_{\mathsf{cl},c}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{cl}}, (S_1, \ldots, S_c)).$$

5. Algorithm $\mathcal{B}_1$ outputs the number of instances 1^c, the circuit size 1^s, and the index $i^* \in [c]$. The challenger responds with $\mathsf{crs}_{\mathsf{BARG}}$. Here s is the size of the circuit $C_{\mathsf{ClauseSAT}}$ that computes $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$ from Fig. 1.

6. Algorithm $\mathcal{B}_1$ constructs the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidBARG}}, \mu_b)$, where $C_{\mathsf{ValidBARG}}$ is the Boolean circuit that takes as input a BARG proof π and outputs $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi)$.

7. Algorithm $\mathcal{B}$ sets $\mathsf{ct} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{ct}_{\mathsf{WE}})$ and outputs $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{ct})$.

If the challenger samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^c, 1^s)$, algorithm $\mathcal{B}$ simulates an execution of $\mathsf{Hyb}_2^{(b)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]$. If the challenger samples $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}^*(1^\lambda, 1^c, 1^s, i^*)$, then algorithm $\mathcal{B}$ simulates an execution of $\mathsf{Hyb}_3^{(b)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]$. We conclude that algorithm $\mathcal{B}$ breaks mode indistinguishability with the same advantage ε. ☐

Lemma 4.7. *If Π_{WE} satisfies semantic security, Π_{SSB} is correct and somewhere statistically binding, Π_{BARG} satisfies somewhere statistical soundness, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_3^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(1)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Suppose $|\Pr[\mathsf{Hyb}_3^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(1)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. First, we note that with probability at least ε, it must be the case that $\mathcal{A}_0$ outputs $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1)$ where for β_i defined according to Eq. (4.1), it holds that $P(\beta_1, \ldots, \beta_K) = 0$. When $P(\beta_1, \ldots, \beta_K)$, the output in both experiments is always 0. Thus, in the subsequent analysis, we assume that $P(\beta_1, \ldots, \beta_K) = 0$. We use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct an efficient (non-uniform) adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ for the mode indistinguishability game. The behavior of $\mathcal{B}_0$ is shown in Fig. 2. On input the non-uniform advice $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, (\beta_1, \ldots, \beta_K))$, algorithm $\mathcal{B}_1$ works as follows:

1. On input the security parameter 1^λ and the advice string $(C, P, (x_1, \ldots, x_K), \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, (\beta_1, \ldots, \beta_K))$, algorithm $\mathcal{B}_1$ outputs 0 if $P(\beta_1, \ldots, \beta_K) = 1$.

2. Otherwise, interpret $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and $P \colon \{0,1\}^K \to \{0,1\}$. Let c be the number of clauses in P, $t \leq K$ be the maximum number of variables that appears in any clause of P, and $S_i \subseteq [K]$ be the variables that appear in clause i of P. Let $i^* \in [c]$ be the first index where for all $j \in S_{i^*}$, $\beta_j = 0$.

3. Algorithm $\mathcal{B}_1$ samples $\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^t, K, S_{i^*})$ and $\mathsf{hk}_{\mathsf{cl}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{cl}}}, 1^1, c, \{i^*\})$.

4. Algorithm $\mathcal{B}_1$ computes

$$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$$
$$(\mathsf{h}_{\mathsf{cl}}, \pi_{\mathsf{cl},1}, \ldots, \pi_{\mathsf{cl},c}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{cl}}, (S_1, \ldots, S_c)).$$

It then computes $\mathsf{crs}_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Setup}^*(1^\lambda, 1^c, 1^s, i^*)$, where s is the size of the circuit $C_{\mathsf{ClauseSAT}}$ that computes $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$ from Fig. 1.

5. Algorithm $\mathcal{B}_1$ gives $C_{\mathsf{ValidBARG}}, \mu_0, \mu_1$ to the challenger and receives a ciphertext $\mathsf{ct}_{\mathsf{WE}}$. Here, $C_{\mathsf{ValidBARG}}$ is the Boolean circuit that takes as input a BARG proof π and outputs $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi)$.

6. Algorithm $\mathcal{B}_1$ sets $\mathsf{ct} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{ct}_{\mathsf{WE}})$ and outputs $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{ct})$.

By construction, algorithm $\mathcal{B}$ simulates $\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}, \mathsf{crs}_{\mathsf{BARG}}$ exactly according to the specification of $\mathsf{Hyb}_3^{(0)}$ and $\mathsf{Hyb}_3^{(1)}$. To complete the proof, we first argue that with overwhelming probability over the choice of $\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}$, and $\mathsf{crs}_{\mathsf{BARG}}$, it holds that for all $\pi \in \{0,1\}^*$, $C_{\mathsf{ValidBARG}}(\pi) = 0$:

- Let $\mathcal{L}_{\mathsf{ClauseSAT}}$ be the language associated with $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$. We first show that $i^* \notin \mathcal{L}_{\mathsf{ClauseSAT}}$. Consider a candidate witness $(S, \pi_{\mathsf{cl}}, j, x, \pi_{\mathsf{inst}}, w)$ for statement i^*.
  - First, $\mathsf{hk}_{\mathsf{cl}}$ is statistically binding on the set $\{i^*\}$, and h_{cl} is a hash of $(S_1, \ldots, S_c)$. Correctness and somewhere statistically binding of Π_{SSB} implies that the only set S for which $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{cl}}, \mathsf{h}_{\mathsf{cl}}, i^*, S, \cdot)$ outputs 1 is $S = S_{i^*}$.
  - Next, $\mathsf{hk}_{\mathsf{inst}}$ is statistically binding on the set S_{i^*} and $\mathsf{h}_{\mathsf{inst}}$ is a hash of $(x_1, \ldots, x_K)$. Correctness and somewhere statistically binding of Π_{SSB} implies that for all $j \in S_{i^*}$, the only values of x for which $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{pk}}, \mathsf{h}_{\mathsf{pk}}, j, x, \cdot)$ outputs 1 is if $x = x_j$.
  - By construction of i^*, for all $j \in S_{i^*}$, we have that $\beta_j = 0$. This means that for all $w \in \{0,1\}^h$, it holds that $C(x_j, w) = 0$. From above, $x = x_j$, so we conclude that $C(x, w) = 0$.

  Taken together, we conclude that $i^* \notin \mathcal{L}_{\mathsf{ClauseSAT}}$.

- Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{BARG}}$ to be statistically sound on $i^* \in [c]$. Since $i^* \notin \mathcal{L}_{\mathsf{ClauseSAT}}$, somewhere statistical soundness of Π_{BARG} states that with overwhelming probability over the choice of $\mathsf{crs}_{\mathsf{BARG}}$, there does not exist a proof π where $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi) = 1$. Correspondingly, this means that there does not exist any π where $C_{\mathsf{ValidBARG}}(\pi) = 1$.

Since $C_{\mathsf{ValidBARG}}(\pi) = 0$ for all π with overwhelming probability, the witness encryption challenger constructs the challenge ciphertext in one of two possible ways:

- If the challenger replies with $\mathsf{ct} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidBARG}}, \mu_0)$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_3^{(0)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_3^{(0)}(\mathcal{A}) = 1]$.

- If the challenger replies with $\mathsf{ct} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidBARG}}, \mu_1)$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_3^{(1)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_3^{(1)}(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ breaks semantic security with advantage $\varepsilon - \mathsf{negl}(\lambda)$. The lemma follows. □

Security now follows by combining Lemmas 4.4 to 4.7. □

Theorem 4.8 (Local Decryption). *Suppose Π_{SSB} is succinct. Then Construction 4.1 supports local decryption.*

Proof. This follows by inspection. We define the preprocessing and local decryption algorithms as follows:

- $\mathsf{Preprocess}(\mathsf{ct}, C, P, (x_1, \ldots, x_K))$: On input the ciphertext $\mathsf{ct} = (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, \mathsf{ct}_{\mathsf{WE}})$, the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, the policy $P\colon \{0,1\}^K \to \{0,1\}$, and the statements $x_1, \ldots, x_K \in \{0,1\}^n$, the preprocessing algorithm computes a hash of the instances $(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$. Then, it outputs the hints $(\mathsf{ht}_1, \ldots, \mathsf{ht}_K)$ where $\mathsf{ht}_i = (x_i, \pi_{\mathsf{inst},i})$

- $\mathsf{DecryptLocal}(\mathsf{ct}, C, P, \{(i, \mathsf{ht}_i, w_i)\}_{i \in S})$: On input the ciphertext ct, the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, the policy $P\colon \{0,1\}^K \to \{0,1\}$, and the hints $\mathsf{ht}_i = (x_i, \pi_{\mathsf{inst},i})$ together with witnesses $w_i \in \{0,1\}^h$ for each $i \in S$, the local decryption algorithm proceeds as follows:
  - For each $i \in [K]$, let $\beta_i = 1$ if $i \in S$ and $\beta_i = 0$ if $i \notin S$. If $P(\beta_1, \ldots, \beta_K) \neq 0$, then output $\perp$.
  - Let c be the number of clauses that appear in P. For $i \in [c]$, let $S_i \subseteq [K]$ denote the variables that appear in clause i of P (under a canonical ordering of the variables). Compute a hash of the clauses $(\mathsf{h}_{\mathsf{cl}}, \pi_{\mathsf{cl},1}, \ldots, \pi_{\mathsf{cl},c}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{cl}}, (S_1, \ldots, S_c))$.
  - Since P is a CNF and $P(\beta_1, \ldots, \beta_K) \neq 0$, every clause of P must contain a satisfied literal. In other words, for every $i \in [c]$, there exists an index $j_i \in S \cap S_i$ where $C(x_{j_i}, w_{j_i}) = 1$. Let j_i be the smallest such index and define $w_{\mathsf{BARG},i} = (S_i, \pi_{\mathsf{cl},i}, j_i, x_{j_i}, \pi_{\mathsf{inst},j_i}, w_{j_i})$.
  - Let $C_{\mathsf{ClauseSAT}}$ be the Boolean circuit that computes the NP relation $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$ from Fig. 1. Construct an index BARG proof
  $$\pi_{\mathsf{BARG}} \leftarrow \mathsf{BARG.Prove}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, (w_{\mathsf{BARG},1}, \ldots, w_{\mathsf{BARG},c})).$$
  - Let $C_{\mathsf{ValidBARG}}$ be the Boolean circuit that takes as input a BARG proof π and outputs
  $$\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, C_{\mathsf{ClauseSAT}}, c, \pi).$$

  Output $\mathsf{WE.Decrypt}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidBARG}}, \pi_{\mathsf{BARG}})$.

By succinctness of Π_{SSB}, the size of the openings $\pi_{\mathsf{inst},i}$ computed by Preprocess has size $\mathsf{poly}(\lambda, n, t, \log K)$. Corresponding the size of the hints output by Preprocess have size $n + \mathsf{poly}(\lambda, n, t, \log K)$, so succinctness follows. Finally, correctness follows by construction (namely, the composition of Preprocess and DecryptLocal coincides with the Decrypt algorithm in Construction 4.1). □

Instantiation. Construction 4.1 yields a succinct witness encryption scheme for CNF policies from plain witness encryption in conjunction with somewhere statistically binding hash functions and somewhere statistically sound (index) BARGs for NP [WW22]. When encrypting to a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a CNF policy $P\colon \{0,1\}^K \to \{0,1\}$ with c clauses and where each clauses has size at most t, the size of the ciphertext in Construction 4.1 is $\mathsf{poly}(\lambda, |C|, t, \log c)$. In particular, the size of the ciphertext scales with the size of a *single* clause and polylogarithmically with the total number of clauses. In particular, when the size of each clause is constant, then the overall ciphertext size is polylogarithmic in the size of the policy. We summarize the efficiency properties of our instantiation in the following corollary:

Corollary 4.9 (Succinct Witness Encryption for CNF Policies)**.** *Let λ be a security parameter. Assuming the existence of somewhere statistically binding hash functions, somewhere statistically sound BARGs for* NP*, and witness encryption for* NP*, there exists a succinct witness encryption scheme for CNF policies (that supports local decryption). An encryption of a message μ with respect to a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and a CNF policy $P\colon \{0,1\}^K \to \{0,1\}$ with c clauses and maximum clause size t has size $|\mu| + \mathsf{poly}(\lambda, |C|, t, \log c)$. In particular, when consider CNFs where each clause contains a constant number of variables (e.g., $t = O(1)$), the ciphertext size scales polylogarithmically with the size of the policy P.*

Remark 4.10 (Conjunctions of Local Monotone Predicates)**.** Construction 4.1 immediately generalizes to yield a succinct witness encryption scheme for conjunctions of arbitrary (local) monotone predicates. Consider a policy $P\colon \{0,1\}^K \to \{0,1\}$ of the form
$$P(\beta_1, \ldots, \beta_K) \coloneqq P_1(\vec{\beta}_{S_1}) \wedge \cdots \wedge P_c(\vec{\beta}_{S_c}),$$
where $P_1, \ldots, P_c$ are arbitrary monotone predicates on the variables $\vec{\beta}_{S_i} \coloneqq (\beta_j)_{j \in S_i}$. A CNF corresponds to the special case where each local predicate P_i is a disjunction on the variables $\vec{\beta}_{S_i}$. To generalize Construction 4.1 to this setting, we proceed as follows:

- We take h_{cl} to be a hash of the pairs $(P_1, S_1), \ldots, (P_c, S_c)$. Namely, h_{cl} now binds to a predicate together with the set of variables on which it depends.
- We modify the NP relation $\mathcal{R}_{\mathsf{ClauseSAT}}$ to check satisfiability of the i^{th} predicate:

  Fixed values: hash keys $\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}$, a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and hashes $\mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}$
  Statement: index $i \in \mathbb{N}$
  Witness: a pair (P, S), an opening π_{cl}, and triples $(x_i, w_i, \pi_{\mathsf{inst},i})_{i \in S}$

  On input a statement $i \in \mathbb{N}$ and a tuple $((P, S), \pi_{\mathsf{cl}}, (x_i, w_i, \pi_{\mathsf{inst},i})_{i \in S})$, output 1 if the following hold:

  - $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{cl}}, \mathsf{h}_{\mathsf{cl}}, i, (P, S), \pi_{\mathsf{cl}}) = 1$;
  - For all $i \in S$, $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x_i, \pi_{\mathsf{inst},i}) = 1$; and
  - $P((\beta_i)_{i \in S}) = 1$, where $\beta_i = C(x_i, w_i)$ for all $i \in S$.

  Figure 3: The modified NP relation $\mathcal{R}_{\mathsf{ClauseSAT}}[\mathsf{hk}_{\mathsf{inst}}, \mathsf{hk}_{\mathsf{cl}}, C, \mathsf{h}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{cl}}]$ to support general predicates

  In the security proof, we use the fact that for every false instance $(C, P, (x_1, \ldots, x_K))$, there always exists an index $i^* \in [c]$ where the local predicate P_{i^*} is unsatisfiable for the statements $x_{S_{i^*}}$ (*irrespective* of the choice of witness). This property critically relies on the assumption that the local predicates are monotone. Indeed, if the local predicates were non-monotone, it could be the case that *every* predicate is locally satisfiable for *some* choice of witness, but the policy is globally unsatisfiable when the adversary is forced to use a *consistent* witness for each instance (across different clauses). Our proof strategy in Theorem 4.3 assumes that there is always one unsatisfiable clause, which will always be the case for a conjunction of monotone predicates.

With these two modifications, we obtain a succinct witness encryption scheme that supports policies that can be represented by a conjunction of local monotone predicates. An encryption of a message μ now with respect to the Boolean relation C and a policy P comprised of local predicates $(P_1, \ldots, P_c)$ has size $|\mu| + \mathsf{poly}(\lambda, |C|, \max_{i \in [c]} |P_i|, \log c)$. Once more, the ciphertext size scales with the size of a *single* predicate and polylogarithmically with the total number of predicates.

4.2 Succinct Witness Encryption for DNF Policies

In this section, we show how to construct a succinct witness encryption scheme for DNF policies by combining a witness encryption scheme together with a function-binding hash function. Our scheme is limited to trapdoor NP relations where there is an efficiently-computable algorithm that can decide membership in the language (given some trapdoor information). Specifically, we define a trapdoor NP relation as follows:

Definition 4.11 (Trapdoor NP Relation)**.** Let $\mathcal{R} = \{\mathcal{R}_n\}_{n \in \mathbb{N}}$ be a family of NP relations where $\mathcal{R}_n \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is an efficiently-computable relation. We say $\mathcal{R}$ is a trapdoor NP relation if there exists an efficiently-computable family of circuits $C = \{C_n\}_{n \in \mathbb{N}}$ such that for all $n \in \mathbb{N}$ and all $x \in \{0,1\}^n$,

$$C_n(x) = 1 \qquad \text{if and only if} \qquad \exists w \in \{0,1\}^h : \mathcal{R}_n(x, w) = 1.$$

Importantly for our applications, the construction itself does not require knowledge of the trapdoor circuits C. Otherwise, the language is in P and witness encryption is trivial. The *existence* of the trapdoor relation is only needed in the security proof (where the circuit family could be provided to the reduction algorithm as non-uniform advice). As we show in Section 6, succinct witness encryption for trapdoor languages suffices for applications to both computational secret sharing and the notion of monotone-policy encryption we introduce in this work.

Construction 4.12 (Succinct Witness Encryption for DNF Policies)**.** Let λ be a security parameter, $\mathcal{M}$ be a message space, and $\mathcal{P}$ be the set of Boolean formulas in disjunctive normal form. Our construction relies on the following:

- Let $\Pi_{\mathsf{LHE}} = (\mathsf{LHE.KeyGen}, \mathsf{LHE.Encrypt}, \mathsf{LHE.Eval}, \mathsf{LHE.Decrypt})$ be a leveled homomorphic encryption scheme. Let $\ell_{\mathsf{ct}} = \ell_{\mathsf{ct}}(\lambda, d_{\max})$ be a bound on the length of the ciphertext (encrypting a single bit) in Π_{LHE} as a function of the security parameter λ and the depth bound $d_{\max}$.
- Let $\Pi_{\mathsf{FBH}} = (\mathsf{FBH.Setup}, \mathsf{FBH.Hash}, \mathsf{FBH.Verify})$ be a function-binding hash function for disjunctions of block functions.
- Let $\Pi_{\mathsf{WE}} = (\mathsf{WE.Encrypt}, \mathsf{WE.Decrypt})$ be a witness encryption scheme with message space $\mathcal{M}$.

Let $\mathcal{R} = \{\mathcal{R}_n\}_{n\in\mathbb{N}}$ be a family of trapdoor NP relations, where the associated family of trapdoor circuits $C_{\mathsf{td}} = \{C_{\mathsf{td},n}\}_{n\in\mathbb{N}}$ can be computed by a circuit of depth at most $d_{\mathsf{td}} = d_{\mathsf{td}}(n)$ and size at most $s_{\mathsf{td}}(n)$. We construct a succinct witness encryption scheme for $\mathcal{R}$ with message space $\mathcal{M}$ and policy space $\mathcal{P}$ as follows:

- $\mathsf{Encrypt}(1^\lambda, C, P, (x_1, \ldots, x_K), \mu)$: On input the security parameter λ, the Boolean relation $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, the Boolean policy $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and the message $\mu \in \mathcal{M}$, the encryption algorithm proceeds as follows:
  - Let c be the number of min-terms in P, and let $t \le K$ be the maximum number of variables that appears in any min-term of P. For any collection of t instances $\tilde{x}_1, \ldots, \tilde{x}_t \in \{0,1\}^n$, let $U_{\tilde{x}_1,\ldots,\tilde{x}_t}(C)$ be the universal circuit that takes as input the description of a circuit C of depth at most d_{td} and size at most s_{td} and outputs
  $$U_{\tilde{x}_1,\ldots,\tilde{x}_t}(C) := \bigwedge_{j\in[t]} C(\tilde{x}_i). \tag{4.2}$$
  Note that we assume the instances $\tilde{x}_1, \ldots, \tilde{x}_t$ are hard-wired in the description of $U_{\tilde{x}_1,\ldots,\tilde{x}_t}$. Let d'_{td} be a bound on the depth of the circuit of $U_{\tilde{x}_1,\ldots,\tilde{x}_t}$ (for an arbitrary choice of $\tilde{x}_1, \ldots, \tilde{x}_t$).
  - Next, for each min-term φ of P over instances $x_{i_{\varphi,1}}, \ldots, x_{i_{\varphi,t}} \in [K]$, we define
  $$U_\varphi(C) := U_{x_{i_{\varphi,1}},\ldots,x_{i_{\varphi,t}}}(C). \tag{4.3}$$
  In the following, we assume that the indices $i_{\varphi,1}, \ldots, i_{\varphi,t}$ of the instances that appear in φ are in lexicographic order, so for every φ, there is a canonical description of the universal circuit U_φ.
  - Sample $(\mathsf{pk}_{\mathsf{LHE}}, \mathsf{sk}_{\mathsf{LHE}}) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'_{\mathsf{td}}})$. Let $C_{\mathsf{dummy}} \colon (\{0,1\}^n)^t \to \{0,1\}$ be the dummy circuit that takes as input t statements and always outputs 1 (and padded to a string of size s_{td}). Compute $\mathsf{ct}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, C_{\mathsf{dummy}})$.
  - Let $d_{\mathsf{dec}} = d_{\mathsf{dec}}(\lambda)$ and $s_{\mathsf{dec}} = s_{\mathsf{dec}}(\lambda)$ be bounds on the depth and size, respectively, of the Boolean circuit that computes $C_{\mathsf{dec}}(\mathsf{ct}) := \mathsf{LHE.Decrypt}(\mathsf{sk}_{\mathsf{LHE}}, \mathsf{ct})$. Sample $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}}, 1^{d_{\mathsf{dec}}}, 1^{s_{\mathsf{dec}}}, c, \bot)$.
  - For each min-term φ_i in P, compute $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}})$.
  - Compute a hash of the ciphertexts $(\mathsf{h}, \pi_{\mathsf{ct},1}, \ldots, \pi_{\mathsf{ct},c}) = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_c))$.
  - Let C_{DNFSat} be the Boolean circuit computing the following functionality:

Fixed values: a hash key hk, a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a hash h, a public key $\mathsf{pk}_{\mathsf{LHE}}$, and a ciphertext $\mathsf{ct}_{\mathsf{LHE}}$

Input: an index $i \in \mathbb{N}$, instances $\tilde{x}_1, \ldots, \tilde{x}_t$, witnesses $\tilde{w}_1, \ldots, \tilde{w}_t$, and an opening π

On input the index $i \in \mathbb{N}$, instances $\tilde{x}_1, \ldots, \tilde{x}_t$, witnesses $\tilde{w}_1, \ldots, \tilde{w}_t$, and an opening π, output 1 if the following conditions hold:

* For all $j \in [t]$, it is the case that $C(\tilde{x}_j, \tilde{w}_j) = 1$.
* Let $U_{\tilde{x}_1,\ldots,\tilde{x}_t}$ be the universal circuit from Eq. (4.2). Compute the ciphertext $\mathsf{ct} = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\tilde{x}_1,\ldots,\tilde{x}_t}, \mathsf{ct}_{\mathsf{LHE}})$. Check that $\mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{h}, i, \mathsf{ct}, \pi) = 1$.

Figure 4: The $C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}]$ circuit

Compute the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}], \mu)$ and output the ciphertext $\mathsf{ct} = (\mathsf{hk}, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{WE}})$.

- $\mathsf{Decrypt}(\mathsf{ct}, C, P, (x_1, \ldots, x_K), (w_1, \ldots, w_K))$: On input a ciphertext $\mathsf{ct} = (\mathsf{hk}, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{WE}})$, a Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a policy $P \in \mathcal{P}$ where $P\colon \{0,1\}^K \to \{0,1\}$, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_K \in \{0,1\}^h$, the decryption algorithm proceeds as follows:
  - Let c be the number of min-terms in P. For each min-term φ_i in P, compute $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}})$, where U_{φ_i} is defined as in Eq. (4.3).
  - Compute a hash of the ciphertexts $(\mathsf{h}, \pi_{\mathsf{ct},1}, \ldots, \pi_{\mathsf{ct},c}) = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_c))$.
  - Since P is satisfied, there must exist an index $i \in [K]$ such that φ_i is satisfied. Let $i_1, \ldots, i_t$ be the variables that appear in φ_i. Let $w = (i, x_{i_1}, \ldots, x_{i_t}, w_{i_1}, \ldots, w_{i_t}, \pi_{\mathsf{ct},i})$. Output $\mathsf{WE.Decrypt}(\mathsf{ct}, w)$.

Theorem 4.13 (Correctness)**.** *If Π_{FBH} and Π_{WE} are correct, then Construction 4.12 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, any Boolean policy $P \in \mathcal{P}$ where $P\colon \{0,1\}^n \to \{0,1\}$, any collection of statements $x_1, \ldots, x_K \in \{0,1\}^n$ and witnesses $w_1, \ldots, w_K \in \{0,1\}^h$ where $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$, and any message $\mu \in \mathcal{M}$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, P, (x_1 \ldots, x_K), \mu)$ and consider $\mathsf{Decrypt}(\mathsf{ct}, C, P, (x_1, \ldots, x_K), (w_1, \ldots, w_K))$:

- By construction, $\mathsf{ct} = (\mathsf{hk}, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{WE}})$, where $(\mathsf{pk}_{\mathsf{LHE}}, \mathsf{sk}_{\mathsf{LHE}}) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'_{\mathsf{td}}})$ and $\mathsf{ct}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, C_{\mathsf{dummy}})$.
- Let c be the number of min-terms in P. For each min-term φ_i in P, both the encryption and the decryption algorithms compute $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}})$. In addition, both algorithms compute a hash $(\mathsf{h}, \pi_{\mathsf{ct},1}, \ldots, \pi_{\mathsf{ct},c}) = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_c))$. By correctness of Π_{FBH}, this means
$$\forall i \in [c] : \mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{h}, i, \mathsf{ct}_i, \pi_{\mathsf{ct},i}) = 1.$$
- Since P is satisfied, there must exist an index $i \in [c]$ such that φ_i is satisfied. Let $i_1, \ldots, i_t$ be the variables that appear in φ_i. This means $C(x_{i_j}, w_{i_j}) = 1$ for all $j \in [t]$. By Eq. (4.3),
$$U_{\varphi_i}(C) := U_{x_{i_1}, \ldots, x_{i_t}}(C).$$
Since homomorphic evaluation is deterministic, this means
$$\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}}) = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{x_{i_1}, \ldots, x_{i_t}}, \mathsf{ct}_{\mathsf{LHE}}).$$
This means that $C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}](i, x_{i_1}, \ldots, x_{i_t}, w_{i_1}, \ldots, w_{i_t}, \pi_{\mathsf{ct},i}) = 1$. The claim now follows by correctness of witness encryption. □

Theorem 4.14. *Suppose Π_{LHE} satisfies perfect correctness and CPA-security, Π_{FBH} satisfies function hiding and statistical disjunction binding, and Π_{WE} satisfies semantic security. Then, Construction 4.12 satisfies semantic security.*

Proof. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an efficient non-uniform adversary for the semantic security game. In particular, on input the security parameter 1^λ, algorithm $\mathcal{A}_0$ outputs a tuple $(C, P, \vec{x}, \mu_0, \mu_1)$ together with some state information $\mathsf{st}_{\mathcal{A}}$ (of polynomial size) and where $\vec{x} = (x_1, \ldots, x_K)$. Algorithm $\mathcal{A}_1$ takes as input the state $\mathsf{st}_{\mathcal{A}}$ and a ciphertext ct and outputs a bit $b' \in \{0,1\}$. For each $i \in [K]$, define the bit β_i as follows:
$$\begin{cases} 1 & \exists w_i \in \{0,1\}^h : C(x_i, w_i) = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

Our reduction algorithm will take the bits $\vec{\beta} = (\beta_1, \ldots, \beta_K)$, $\mathsf{st}_{\mathcal{A}}$, and the trapdoor circuits $C_{\mathsf{td},n}$ associated with the NP relation defined by C as non-uniform advice. We now define a sequence of hybrid experiments parameterized by a bit $b \in \{0,1\}$:

- $\mathsf{Hyb}_0^{(b)}$: This is the semantic security game with adversary $\mathcal{A}$ and bit $b \in \{0,1\}$:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}_0(1^\lambda)$ outputs $(C, P, \vec{x}, \mu_0, \mu_1)$ and $\mathsf{st}_{\mathcal{A}}$. For each $i \in [K]$, define the bits β_i according to Eq. (4.4). Let $\vec{\beta} = (\beta_1, \ldots, \beta_K)$.
  - If $P(\vec{\beta}) = 1$, the challenger outputs 0. Otherwise, the challenger invokes $b' \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, \mathsf{ct})$ where $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, C, P, \vec{x}, \mu_b)$. Specifically, the challenger constructs ct as follows:
    - Sample $(\mathsf{pk}_{\mathsf{LHE}}, \mathsf{sk}_{\mathsf{LHE}}) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'_{\mathsf{td}}})$ and compute $\mathsf{ct}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, C_{\mathsf{dummy}})$.
    - Sample $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}}, 1^{d_{\mathsf{dec}}}, 1^{s_{\mathsf{dec}}}, c, \perp)$.
    - For each min-term φ_i in P, compute $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}})$. Compute a hash of the ciphertexts $(\mathsf{h}, \pi_{\mathsf{ct},1}, \ldots, \pi_{\mathsf{ct},c}) = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_c))$.
    - Compute the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}], \mu_b)$ where C_{DNFSat} is the circuit in Fig. 4.
    - Finally, the challenger sets $\mathsf{ct} = (\mathsf{hk}, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{WE}})$.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the output of the experiment.

- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$ except the challenger samples $\mathsf{ct}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, C_{\mathsf{td},n})$, where $C_{\mathsf{td},n}$ is the trapdoor circuit associated with the Boolean relation defined by C.

- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$ except the challenger samples $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}}, 1^{d_{\mathsf{dec}}}, 1^{s_{\mathsf{dec}}}, c, C_{\mathsf{dec}})$ where $C_{\mathsf{dec}}(\mathsf{ct}) := \mathsf{LHE.Decrypt}(\mathsf{sk}_{\mathsf{LHE}}, \mathsf{ct})$.

We write $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ to denote the output distribution of an adversary $\mathcal{A}$ in experiment $\mathsf{Hyb}_i^{(b)}$. In our reduction algorithms below, we will construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$. In all cases, algorithm $\mathcal{B}_0$ behaves as follows:

On input the security parameter $\lambda \in \mathbb{N}$:

- Run $(C, P, \vec{x}, \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_0(1^\lambda)$.
- For each $i \in [K]$, compute β_i according to Eq. (4.1). Let $\vec{\beta} = (\beta_1, \ldots, \beta_K)$.
- Output $(C, P, \vec{x}, \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, C_{\mathsf{td},n}, d_{\mathsf{td}'})$, where $C_{\mathsf{td},n}$ is the trapdoor circuit associated with the Boolean relation defined by C and $d_{\mathsf{td}'}$ is a bound on the depth of the circuit $U_{\tilde{x}_1,\ldots,\tilde{x}_t}$ from Eq. (4.2).

Figure 5: The pre-processing algorithm $\mathcal{B}_0$

We now analyze each adjacent pair of hybrid experiments.

Lemma 4.15. *If Π_{LHE} satisfies CPA-security, then for all $b \in \{0,1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Take any $b \in \{0,1\}$ and suppose $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ for the CPA-security game:

1. On input the security parameter and the advice string $(C, P, \vec{x}, \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, C_{\mathsf{td},n})$, algorithm $\mathcal{B}_1$ outputs 0 if $P(\vec{\beta}) = 0$. Otherwise, algorithm $\mathcal{B}_1$ outputs the depth bound $1^{d'_{\mathsf{td}}}$. The challenger replies with $\mathsf{pk}_{\mathsf{LHE}}$.

2. Algorithm $\mathcal{B}_1$ outputs the challenge messages C_{dummy} and $C_{\mathsf{td},n}$ and receives a challenge ciphertext $\mathsf{ct}_{\mathsf{LHE}}$.

3. Algorithm $\mathcal{B}_1$ samples $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}}, 1^{d_{\mathsf{dec}}}, 1^{s_{\mathsf{dec}}}, c, \perp)$.

4. For each min-term φ_i in P, algorithm $\mathcal{B}_1$ computes $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}})$. Next, it computes a hash of the ciphertexts $(\mathsf{h}, \pi_{\mathsf{ct},1}, \ldots, \pi_{\mathsf{ct},c}) = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_c))$.

5. Compute the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}], \mu_b)$.

6. Algorithm $\mathcal{B}_1$ defines $\mathsf{ct} = (\mathsf{hk}, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{WE}})$ and outputs $\mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, \mathsf{ct})$.

By definition, the challenger samples $(\mathsf{pk}_{\mathsf{LHE}}, \mathsf{sk}_{\mathsf{LHE}}) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'_{\mathsf{td}}})$, so the public key $\mathsf{pk}_{\mathsf{LHE}}$ is distributed as in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$. Next, if the challenger computes $\mathsf{ct}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, C_{\mathsf{dummy}})$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_0^{(b)}$. If $\mathsf{ct}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, C_{\mathsf{td},n})$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(b)}$. We conclude that algorithm $\mathcal{B}$ breaks CPA-security with the same advantage ε. □

Lemma 4.16. *If Π_{FBH} satisfies function hiding, then for all $b \in \{0,1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Take any $b \in \{0,1\}$ and suppose $|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ for the function-hiding game.

1. On input the security parameter 1^λ and advice string $(C, P, \vec{x}, \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, C_{\mathsf{td},n})$, algorithm $\mathcal{B}_1$ samples a key-pair $(\mathsf{pk}_{\mathsf{LHE}}, \mathsf{sk}_{\mathsf{LHE}}) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'_{\mathsf{td}}})$ and constructs the ciphertext $\mathsf{ct}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, C_{\mathsf{td},n})$:

2. Let ℓ_{ct} be the length of the ciphertext output by $\mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, \cdot)$. Let d_{dec} and s_{dec} be a bound on the depth and size of the Boolean circuit that computes the decryption circuit C_{dec}. Algorithm $\mathcal{B}_1$ outputs the parameters $1^{\ell_{\mathsf{ct}}}$, $1^{d_{\mathsf{dec}}}$, and $1^{s_{\mathsf{dec}}}$, the number of min-terms c, together with the decryption circuit $C_{\mathsf{dec}}(\mathsf{ct}) \coloneqq \mathsf{LHE.Decrypt}(\mathsf{sk}_{\mathsf{LHE}}, \mathsf{ct})$. The challenger replies with a hash key hk.

3. For each min-term φ_i in P, algorithm $\mathcal{B}_1$ computes $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}})$. Next, it computes a hash of the ciphertexts $(\mathsf{h}, \pi_{\mathsf{ct},1}, \ldots, \pi_{\mathsf{ct},c}) = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_c))$.

4. Compute the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}], \mu_b)$.

5. Algorithm $\mathcal{B}_1$ defines $\mathsf{ct} = (\mathsf{hk}, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{WE}})$ and outputs $\mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, \mathsf{ct})$.

If the challenger samples $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}}, 1^{d_{\mathsf{dec}}}, 1^{s_{\mathsf{dec}}}, c, \bot)$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(b)}$. Alternatively, if the challenger sampled $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}}, 1^{d_{\mathsf{dec}}}, 1^{s_{\mathsf{dec}}}, c, C_{\mathsf{td},n})$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_2^{(b)}$. We conclude that algorithm $\mathcal{B}$ breaks function hiding of Π_{FBH} with the advantage ε. □

Lemma 4.17. *If Π_{WE} satisfies semantic security, Π_{FBH} satisfies statistical disjunction binding and Π_{LHE} is perfectly correct, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(1)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $|\Pr[\mathsf{Hyb}_2^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(1)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ for the semantic security game:

1. On input the security parameter 1^λ and advice string $(C, P, \vec{x}, \mu_0, \mu_1, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, C_{\mathsf{td},n})$, algorithm $\mathcal{B}_1$ samples a key-pair $(\mathsf{pk}_{\mathsf{LHE}}, \mathsf{sk}_{\mathsf{LHE}}) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'_{\mathsf{td}}})$ and constructs the ciphertext $\mathsf{ct}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_{\mathsf{LHE}}, C_{\mathsf{td},n})$.

2. Next, algorithm $\mathcal{B}_1$ samples a hash key $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, 1^{\ell_{\mathsf{ct}}}, 1^{d_{\mathsf{dec}}}, 1^{s_{\mathsf{dec}}}, c, C_{\mathsf{dec}})$ where the circuit $C_{\mathsf{dec}}(\mathsf{ct}) \coloneqq \mathsf{LHE.Decrypt}(\mathsf{sk}_{\mathsf{LHE}}, \mathsf{ct})$ is the LHE decryption circuit with $\mathsf{sk}_{\mathsf{LHE}}$ hard-coded.

3. For each min-term φ_i in P, algorithm $\mathcal{B}_1$ computes $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}})$. Next, it computes a hash of the ciphertexts $(\mathsf{h}, \pi_{\mathsf{ct},1}, \ldots, \pi_{\mathsf{ct},c}) = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_c))$.

4. Algorithm $\mathcal{B}_1$ outputs the circuit $C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}]$ together with messages μ_0, μ_1. The challenger replies with a ciphertext $\mathsf{ct}_{\mathsf{WE}}$.

5. Algorithm $\mathcal{B}_1$ defines $\mathsf{ct} = (\mathsf{hk}, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{WE}})$ and outputs $\mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, \mathsf{ct})$.

First, we show that with overwhelming probability over the choice of hk, for all $w = (i, \tilde{x}_1, \ldots, \tilde{x}_t, \tilde{w}_1, \ldots, \tilde{w}_t, \pi)$, it holds that $C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}](w) = 0$.

- First, for all $j \in [t]$, we have that $C(\tilde{x}_j, \tilde{w}_j) = 1$. Otherwise, the output of C_{DNFSat} is already 0.
- Let $\mathsf{ct}' = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\tilde{x}_1,\ldots,\tilde{x}_t}, \mathsf{ct}_{\mathsf{LHE}})$ be the ciphertext computed by C_{DNFSat}. From the previous step, for all $j \in [t]$, there exists $\tilde{w}_j$ where $C(\tilde{x}_j, \tilde{w}_j) = 1$. By the first property of Definition 4.11, this means $C_{\mathsf{td},n}(\tilde{x}_j) = 1$ for all $j \in [t]$. By definition of U_φ (see Eq. (4.2)), this means that
$$U_{\tilde{x}_1,\ldots,\tilde{x}_t}(C_{\mathsf{td},n}) = \bigwedge_{j \in [t]} C_{\mathsf{td},n}(\tilde{x}_j) = 1.$$
- Consider the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_c$ that are hashed to obtain h. By definition, $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U_{\varphi_i}, \mathsf{ct}_{\mathsf{LHE}})$. Let $x_{i_{\varphi,1}}, \ldots, x_{i_{\varphi,t}} \in [K]$ be the instances associated with the i^{th} min-term φ_i of P. Since $P(\vec{\beta}) = 0$ (i.e., the policy is not satisfied), there must exist some index $j \in [t]$ such that for all $w \in \{0,1\}^h$, it holds that $C(x_{i_{\varphi,j}}, w) = 0$. This means that $C_{\mathsf{td},n}(x_{i_{\varphi,j}}) = 0$. By definition of U_φ (see Eqs. (4.2) and (4.3)), this means that
$$U_\varphi(C_{\mathsf{td},n}) = \bigwedge_{j \in [t]} C_{\mathsf{td},n}(x_{i_{\varphi,j}}) = 0.$$
Since $\mathsf{ct}_{\mathsf{LHE}}$ is an encryption of $C_{\mathsf{td},n}$, we appeal to perfect correctness of Π_{LHE} to conclude that for all $i \in [c]$,
$$\mathsf{LHE.Decrypt}(\mathsf{sk}_{\mathsf{LHE}}, \mathsf{ct}_i) = U_{\varphi_i}(C_{\mathsf{td},n}) = 0.$$
- Since hk is statistically disjunction binding with respect to the function C_{dec}, $C_{\mathsf{dec}}(\mathsf{ct}_i) = 0$ for all $i \in [c]$, $C_{\mathsf{dec}}(\mathsf{ct}') = 1$, and the hash h is obtained by evaluating $\mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_c))$, the probability that there exists an index $i \in [t]$ and an opening π where $\mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{h}, i, \mathsf{ct}', \pi) = 1$ is negligible (over the choice of hk).

In particular, with overwhelming probability, $w = (i, \tilde{x}_1, \ldots, \tilde{x}_t, \tilde{w}_1, \tilde{w}_t, \pi)$ is not a valid witness. By construction, $\mathsf{hk}, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}$ are distributed exactly according to the specification of $\mathsf{Hyb}_2^{(0)}$ and $\mathsf{Hyb}_2^{(1)}$. It suffices to consider the distribution of $\mathsf{ct}_{\mathsf{WE}}$ returned by the challenger:

- If $\mathsf{ct} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}], \mu_0)$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_2^{(0)}$.
- If $\mathsf{ct} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}], \mu_1)$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_2^{(1)}$.

We conclude that algorithm $\mathcal{B}$ breaks semantic security with advantage $\varepsilon - \mathsf{negl}(\lambda)$ and the lemma follows. □

Semantic security now follows by combining Lemmas 4.15 to 4.17. □

Instantiation. Construction 4.12 yields a succinct witness encryption scheme for DNF policies from plain witness encryption in conjunction with a leveled homomorphic encryption scheme and a function-binding hash function (for disjunction of block functions). The latter primitives can be built from standard lattice assumptions (e.g., [BV11, FWW23]). When encrypting to a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a DNF policy $P\colon \{0,1\}^K \to \{0,1\}$ with c min-terms, each of size at most t, the size of the ciphertext in Construction 4.12 is $\mathsf{poly}(\lambda, |C|, t, \log c)$. Similar to the case with CNFs (Construction 4.1 and Corollary 4.9), the size of the ciphertext scales with the size of a single min-term and polylogarithmically with the number of min-terms. When each min-term is over a constant number of variables, then the overall ciphertext is polylogarithmic in the policy size. We summarize our instantiation with the following corollary:

Corollary 4.18 (Succinct Witness Encryption for DNF Policies). *Let λ be a security parameter. Assuming the existence of leveled homomorphic encryption, function-binding hash functions for disjunction of block functions, and witness encryption for* NP*, there exists a succinct witness encryption scheme for DNF policies over a* trapdoor NP *relation. An encryption of a message μ with respect to a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and a DNF policy $P\colon \{0,1\}^K \to \{0,1\}$ with c min-terms of size at most t is $|\mu| + \mathsf{poly}(\lambda, |C|, t, \log c)$. When each min-term is over a constant number of variables (e.g., $t = O(1)$), then the ciphertext size scales polylogarithmic with the size of P.*

Remark 4.19 (Disjunction of Local Monotone Predicates)**.** Much like the case for Construction 4.1 (see Remark 4.10), Construction 4.12 readily generalizes to yield a succinct witness encryption scheme for disjunctions of arbitrary (local) monotone predicates. Consider a policy $P\colon \{0,1\}^K \to \{0,1\}$ of the form

$$P(\beta_1, \ldots, \beta_K) \coloneqq P_1(\vec{\beta}_{S_1}) \vee \cdots \vee P_c(\vec{\beta}_{S_c})$$

where $P_1, \ldots, P_c$ are arbitrary monotone predicates on the variables $\vec{\beta}_{S_i} \coloneqq (\beta_j)_{j \in S_i}$. A DNF corresponds to the special case where each local predicate P_i is a conjunction over $\vec{\beta}_{S_i}$. To generalize Construction 4.12 to this setting, it suffices to modify the scheme as follows:

- For each $i \in [c]$, let $S_i = \{j_{i,1}, \ldots, j_{i,t}\} \subseteq [K]$. We replace the universal circuit U_{φ_i} from Eqs. (4.2) and (4.3) with the circuit
$$U[P_i, x_{j_{i,1}}, \ldots, x_{j_{i,t}}](C) \coloneqq P_i\big(C(x_{j_{i,1}}), \ldots, C(x_{j_{i,t}})\big), \tag{4.5}$$
where the instances $x_{j_{i,1}}, \ldots, x_{j_{i,t}}$ and the predicate P_i is hard-coded into the description of $U[P_i, x_{j_{i,1}}, \ldots, x_{j_{i,t}}]$. As usual, we assume there is a canonical description of $U[P_i, x_{j_{i,1}}, \ldots, x_{j_{i,t}}]$ that can be derived from P and the instances $x_{j_{i,1}}, \ldots, x_{j_{i,t}}$.

- During encryption and decryption, the ciphertexts ct_i are now computed as
$$\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U[P_i, x_{j_{i,1}}, \ldots, x_{j_{i,t}}], \mathsf{ct}_{\mathsf{LHE}}).$$

- Finally, we modify the relation C_{DNFSat} to be the Boolean circuit that additionally take the local predicate P_i as input:

Fixed values: a hash key hk, a circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a hash h, a public key $\mathsf{pk}_{\mathsf{LHE}}$, and a ciphertext $\mathsf{ct}_{\mathsf{LHE}}$

Input: an index i, a predicate P, instances $\tilde{x}_1, \ldots, \tilde{x}_t$, witnesses $\tilde{w}_1, \ldots, \tilde{w}_t$, and an opening π

On input an index i, a predicate P, instances $\tilde{x}_1, \ldots, \tilde{x}_t$, witnesses $\tilde{w}_1, \ldots, \tilde{w}_t$, and an opening π, output 1 if the following conditions hold:

- $P(C(\tilde{x}_1, \tilde{w}_1), \ldots, C(\tilde{x}_t, \tilde{w}_t)) = 1$.
- Let U_P be the universal circuit from Eq. (4.5). Compute the ciphertext $\mathsf{ct} = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U[P, \tilde{x}_1, \ldots, \tilde{x}_t], \mathsf{ct}_{\mathsf{LHE}})$ and check that $\mathsf{FBH.Verify}(\mathsf{hk}, \mathsf{h}, i, \mathsf{ct}, \pi) = 1$.

Figure 6: The modified $C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}]$ circuit to support general predicates

The security proof follows an identical structure as the proof of Theorem 4.14. The only modification is in the analysis of the final hybrid (i.e., in the analog of Lemma 4.17). Consider the argument in the proof of Lemma 4.17. To rely on witness encryption security in this step, we need to show that for all $w = (i, P, \tilde{x}_1, \ldots, \tilde{x}_t, \tilde{w}_1, \ldots, \tilde{w}_t)$, it is the case that $C_{\mathsf{DNFSat}}[\mathsf{hk}, C, \mathsf{h}, \mathsf{pk}_{\mathsf{LHE}}, \mathsf{ct}_{\mathsf{LHE}}](w) = 0$. We proceed using a similar structure as in the proof of Lemma 4.17:

- First, we have that $P(C(\tilde{x}_1, \tilde{w}_1), \ldots, (\tilde{x}_t, \tilde{w}_t)) = 1$. Otherwise, C_{DNFSat} already outputs 0.

- Let $\mathsf{ct}' = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U[P, \tilde{x}_1, \ldots, \tilde{x}_t], \mathsf{ct}_{\mathsf{LHE}})$ be the ciphertext computed by C_{DNFSat}. For each $i \in [t]$, let $\beta_i = C(\tilde{x}_i, \tilde{w}_i) = 1$. From the previous point, we have that $C(\beta_1, \ldots, \beta_t) = 1$. Let $\beta_i' = C_{\mathsf{td},n}(\tilde{x}_i)$. By definition, whenever $\beta_i = 1$, the instance $\tilde{x}_i$ is true, so $C_{\mathsf{td},n}(\tilde{x}_i) = 1$. Thus, for all $i \in [t]$, we have that $\beta_i' \geq \beta_i$. Since P is monotone, this means that $P(\beta_1', \ldots, \beta_t') \geq P(\beta_1, \ldots, \beta_t) = 1$. From Eq. (4.5), this means that
$$U[P, \tilde{x}_1, \ldots, \tilde{x}_t](C_{\mathsf{td},n}) = P(\beta_1', \ldots, \beta_t') = 1.$$
Since $\mathsf{ct}_{\mathsf{LHE}}$ is an encryption of $C_{\mathsf{td},n}$, by correctness of Π_{LHE}, this means that ct' is an encryption of 1 and
$$C_{\mathsf{dec}}(\mathsf{ct}') = \mathsf{LHE.Decrypt}(\mathsf{sk}_{\mathsf{LHE}}, \mathsf{ct}') = U[P, \tilde{x}_1, \ldots, \tilde{x}_t](C_{\mathsf{td},n}) = 1.$$

- Consider the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_c$ that are hashed by the encryption algorithm to obtain h. By definition, $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\mathsf{LHE}}, U[P_i, x_{j_{i,1}}, \ldots, x_{j_{i,t}}], \mathsf{ct}_{\mathsf{LHE}})$. For each $d \in [t]$, let $\beta_{i,d} = 1$ if there exists $w_{j_{i,d}}$ such that $C(x_{j_{i,d}}, w_{j_{i,d}}) = 1$ and 0 otherwise. Since $(x_1, \ldots, x_K)$ is not a satisfying set of instances with respect to C and P, it follows that $P(\beta_{i,1}, \ldots, \beta_{i,t}) = 0$. Now let $\beta'_{i,d} = C_{\mathsf{td},n}(x_{j_{i,d}})$. If $\beta_{i,d} = 0$, then the instance $x_{j_{i,d}}$ is false, so $\beta'_{i,d} = C_{\mathsf{td},n}(x_{j_{i,d}}) = 0$. This means that $\beta'_{i,d} \le \beta_{i,d}$ for all $d \in [t]$. Since P is monotone, this means
$$P(\beta'_{i,1}, \ldots, \beta'_{i,t}) \le P(\beta_{i,1}, \ldots, \beta_{i,t}) = 0.$$
From Eq. (4.5), this means that
$$U[P_i, x_{j_{i,1}}, \ldots, x_{j_{i,t}}](C_{\mathsf{td},n}) = P_i(\beta'_{i,1}, \ldots, \beta'_{i,t}) = 0.$$
Again by correctness of Π_{LHE}, this means that for all $i \in [c]$,
$$C_{\mathsf{dec}}(\mathsf{ct}_i) = \mathsf{LHE.Decrypt}(\mathsf{sk}_{\mathsf{LHE}}, \mathsf{ct}_i) = U[P_i, x_{j_{i,1}}, \ldots, x_{j_{i,t}}](C_{\mathsf{td},n}) = 0.$$

- The claim now follows by the fact that hk is statistically disjunction binding with respect to the function C_{dec}, exactly as in the proof of Lemma 4.17.

Taken altogether, we obtain a succinct witness encryption scheme for trapdoor NP relations that supports policies that can be represented by a disjunction of local monotone predicates. An encryption of a message μ now with respect to the Boolean relation C and a policy P comprised of local predicates $(P_1, \ldots, P_c)$ has size $|\mu| + \mathsf{poly}(\lambda, |C|, \max_{i \in [c]} |P_i|, \log c)$. Once more, the ciphertext size scales with the size of a *single* predicate and polylogarithmically with the total number of predicates.

5 Succinct Unique Witness Map for Read-Once Bounded-Space Policies

In this section, we show how to construct a succinct unique witness map for batch NP languages [CPW20]. Unique witness maps can also be viewed as a publicly-verifiable witness PRF [Zha16, CPW23]. As discussed in Section 1 (see also Remarks 5.2 and 5.3), a succinct unique witness map for batch NP immediately implies a succinct witness encryption for batch NP as well as a SNARG for monotone-policy batch NP (relative to the same policy class). We begin with the definition.

Definition 5.1 (Succinct Unique Witness Map for Batch Languages)**.** Let $\mathcal{P}$ be a family of policies. A succinct unique witness map for batch NP with policy family $\mathcal{P}$ is a tuple of efficient algorithms $\Pi_{\mathsf{UWM}} = (\mathsf{Setup}, \mathsf{Map}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, C, K) \to \mathsf{crs}$: On input the security parameter $1^\lambda \in \mathbb{N}$, a Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and a bound on the number of instances K, the setup algorithm outputs a common reference string crs.
- $\mathsf{Map}(\mathsf{crs}, P, (x_1, \ldots, x_K), (w_1, \ldots, w_K)) \to \sigma$: On input the common reference string crs, a Boolean policy $P \in \mathcal{P}$ where $P\colon \{0,1\}^K \to \{0,1\}$, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_K \in \{0,1\}^h$, the mapping algorithm *deterministically* outputs a canonical witness σ.
- $\mathsf{Verify}(\mathsf{crs}, P, (x_1, \ldots, x_K), \sigma) \to b$: On input the common reference string crs, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and a canonical witness σ, the deterministic verification algorithm outputs a bit $b \in \{0,1\}$.

Moreover, we require that Π_{UWM} satisfy the following properties:

- **Completeness:** For all $\lambda \in \mathbb{N}$, all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all Boolean policies $P \in \mathcal{P}$ where $P\colon \{0,1\}^K \to \{0,1\}$, and all tuples of instances $\vec{x} = (x_1, \ldots, x_K)$ and witnesses $\vec{w} = (w_1, \ldots, w_K)$ where $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$, we have that
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, P, \vec{x}, \sigma) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C, K) \\ \sigma = \mathsf{Map}(\mathsf{crs}, P, \vec{x}, \vec{w}) \end{array}\right] = 1.$$

- **Uniqueness:** For all $\lambda \in \mathbb{N}$, all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all Boolean policies $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, all tuples of instances $\vec{x} = (x_1, \ldots, x_K)$, all tuples of witnesses $\vec{w} = (w_1, \ldots, w_K)$, $\vec{w}' = (w_1', \ldots, w_K')$ where $P\big(C(x_1, w_1), \ldots, C(x_K, w_K)\big) = P\big(C(x_1, w_1'), \ldots, C(x_K, w_K')\big) = 1$, we have that

$$\Pr\Big[\mathsf{Map}(\mathsf{crs}, P, \vec{x}, \vec{w}) = \mathsf{Map}(\mathsf{crs}, P, \vec{x}, \vec{w}') : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C, K)\Big] = 1.$$

- **Selective soundness:** For a security parameter $\lambda \in \mathbb{N}$ and an adversary $\mathcal{A}$, we define the selective soundness game as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a Boolean policy $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, and challenge instances $x_1^*, \ldots, x_K^* \in \{0,1\}^n$.
  - If there exists $w_1, \ldots, w_K \in \{0,1\}^h$ such that $P(C(x_1^*, w_1), \ldots, C(x_K^*, w_K)) = 1$, then the challenger outputs 0. Otherwise, the challenger sends $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C, K)$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs σ.
  - The output of the experiment is the bit $b' = \mathsf{Verify}\big(\mathsf{crs}, (x_1^*, \ldots, x_K^*), \sigma\big)$.

  The unique witness map satisfies selective soundness if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b' = 1] = \mathsf{negl}(\lambda)$ in the selective soundness game.

- **Succinctness:** There exists universal polynomials $\mathsf{poly}_1, \mathsf{poly}_2$ such that for all $\lambda \in \mathbb{N}$, all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all Boolean policies $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, all tuples of instances $\vec{x} = (x_1, \ldots, x_K)$, all tuples of witnesses $\vec{w} = (w_1, \ldots, w_K)$ where $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$, the following properties hold:
  - The size of the reference string crs output by $\mathsf{Setup}(1^\lambda, C, K)$ satisfies $|\mathsf{crs}| \le o(|P|) \cdot \mathsf{poly}_1(\lambda, |C|, \log K)$.
  - The size of the canonical witness σ output by $\mathsf{Map}(\mathsf{crs}, \vec{x}, \vec{w})$ satisfies $|\sigma| \le \mathsf{poly}_2(\lambda, \log K)$.

Remark 5.2 (Succinct Witness Encryption)**.** The work of [CPW20] shows how to obtain witness encryption by composing a unique witness map with a hardcore predicate. The same transformation extends to the batch setting (and preserves succinctness). This means a unique witness map for batch NP and policy family $\mathcal{P}$ (satisfying Definition 5.1) directly implies a succinct witness encryption for batch languages with the same policy family $\mathcal{P}$.

To briefly recall the [CPW20] approach, their transformation constructs witness encryption for an NP language $\mathcal{L}$ from a unique witness map for the OR-language $\mathcal{L} \vee \mathcal{L}'$, where $\mathcal{L}' = \{y \mid \exists\, z : y = \mathsf{PRG}(z)\}$ and PRG denotes a pseudorandom generator. Here, a pair $(x, y) \in \mathcal{L} \vee \mathcal{L}'$ if $x \in \mathcal{L}$ or $y \in \mathcal{L}'$.

- **Encryption:** To encrypt a bit $\mu \in \{0,1\}$ to a statement x, the encrypter samples a random PRG seed z and computes $y = \mathsf{PRG}(z)$. Then, it computes the canonical witness σ for the statement $(x, y) \in \mathcal{L} \vee \mathcal{L}'$ using the witness $(\perp, z)$. Finally, it computes a hardcore bit β of σ and uses that to blind the message μ. The ciphertext is the pair $(y, \beta \oplus \mu)$.

- **Decryption:** Given a witness w for $x \in \mathcal{L}$, the decrypter can compute the same canonical witness σ for $(x, y) \in \mathcal{L} \vee \mathcal{L}'$ using the witness $(w, \perp)$. From here, it can recover the blinding factor β and then the message μ.

To prove security, [CPW20] first replaces y with a uniformly random string, which disables the "trapdoor" branch. Now, if x is a false statement, then $(x, y) \notin \mathcal{L} \vee \mathcal{L}'$. Security of the unique witness map now asserts that the canonical witness σ for (x, y) is unpredictable. Since the bit β is hard-core, this suffices to blind the message μ. Finally, we observe that in the case case of batch languages, this transformation preserves succinctness in the number of instances being batched. Thus, we note that a succinct unique witness map for batch NP with policy family $\mathcal{P}$ as defined above implies succinct witness encryption for batch NP with policy family $\mathcal{P}$ as defined in Definition 3.1.

Remark 5.3 (Non-Adaptive SNARG for Monotone-Policy Batch NP)**.** By definition, any unique witness map for (batch) NP is also a non-interactive argument for (batch) NP (namely, the proof for a statement x is simply the unique witness w associated with x). Thus, a succinct unique witness map for batch NP with policy family $\mathcal{P}$ immediately gives a succinct non-interactive argument (SNARG) for batch NP with policy family $\mathcal{P}$ (also called a "fully-succinct" batch argument [GSWW22, DWW24]). The size of the common reference string for the corresponding SNARG is precisely the size of the reference string for the unique witness map. Selective security for the unique witness map then corresponds to non-adaptive soundness for the resulting SNARG. A construction satisfying the succinctness properties in Definition 5.1 gives a fully succinct monotone-policy BARG where the size of the CRS is sublinear in the policy size.

Construction. In this section, we show how to use indistinguishability obfuscation (together with a somewhere statistically binding hash function and an injective PRG) to construct a succinct unique witness map for monotone policies that can be implemented by a read-once bounded-space Turing machine. In particular, this captures policies like weighted thresholds.

Definition 5.4 (Monotone Read-Once Bounded-Space Policy)**.** Let $P\colon \{0,1\}^K \to \{0,1\}$ be a monotone policy. We say that P can be computed by a read-once Turing machine Γ with S bits of space if there exists a tuple $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$ with the following properties:

- $\mathsf{Step}_i\colon \{0,1\}^S \times \{0,1\} \to \{0,1\}^S$ is a Boolean circuit that implements the i^{th} step of the Turing machine evaluation;
- $c_{\mathsf{init}} \in \{0,1\}^S$ is the initial configuration; and
- $c_{\mathsf{acc}} \in \{0,1\}^S$ is the accepting configuration.

Moreover, $P(\vec{\beta}) = 1$ if and only if $c_K = c_{\mathsf{acc}}$ where we define $c_0 = c_{\mathsf{init}}$ and for all $i \in [K]$, $c_i = \mathsf{Step}_i(c_{i-1}, \beta_i)$.

Reachable states for read-once Turing machines. Our construction of a succinct unique witness map will rely on a notion of the set of reachable states for a given input. Specifically, for a string $\vec{\beta} = (\beta_1, \ldots, \beta_K) \in \{0,1\}^K$ and an index $i \in [K]$, we associate a set of states $T_{\vec{\beta},i} \subseteq \{0,1\}^S$ that are *potentially* reachable after i evaluation steps. In our model, if $\beta_i = 0$, then the evaluator must use the value 0 in position i, but if $\beta_i = 1$, then the evaluator can choose *either* the value 0 *or* the value 1 as its input in position i. In the following, we will sometimes say that $\vec{\beta}' \in \{0,1\}^K$ is "consistent" with $\vec{\beta}$ if for all $i \in [K]$ where $\vec{\beta}_i = 0$, we also have $\vec{\beta}'_i = 0$. We now give the precise characterization of reachable states that we use in our analysis:

Definition 5.5 (Reachable States for a Read-Once Turing Machines)**.** Let $P\colon \{0,1\}^K \to \{0,1\}$ be a monotone policy that can be computed by a read-once Turing machine $(\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$ with space S. We say an ensemble of sets $\{T_{\vec{\beta},i}\}_{\vec{\beta}\in\{0,1\}^K, i\in[0,K]}$ represents an admissible set of reachable states if it satisfies the following properties:

- For all $\vec{\beta} \in \{0,1\}^K$, $c_{\mathsf{init}} \in T_{\vec{\beta},0}$.
- For all $\vec{\beta} \in \{0,1\}^K$, $i \in [K]$, and $c \in \{0,1\}^S$ where $c \in T_{\vec{\beta},i-1}$, we have $\mathsf{Step}_i(c, 0) \in T_{\vec{\beta},i}$.
- For all $\vec{\beta} \in \{0,1\}^K$, $i \in [K]$, and $c \in \{0,1\}^S$ where $c \in T_{\vec{\beta},i-1}$ and $\beta_i = 1$, we have $\mathsf{Step}_i(c, 1) \in T_{\vec{\beta},i}$.
- For all $\vec{\beta} \in \{0,1\}^K$ where $P(\vec{\beta}) = 0$, it holds that $c_{\mathsf{acc}} \notin T_{\vec{\beta},K}$.

We also associate a Boolean function $\mathsf{Reachable}_{\vec{\beta},i}\colon \{0,1\}^S \to \{0,1\}$ with each set $T_{\vec{\beta},i}$ where $\mathsf{Reachable}_{\vec{\beta},i}(c) = 1$ if and only if $c \in T_{\vec{\beta},i}$. In some settings of interest (e.g., weighted thresholds; see Remark 5.7), the Boolean circuit computing $\mathsf{Reachable}_{\vec{\beta},i}(c)$ has a much more compact description than enumerating the elements of $T_{\vec{\beta},i}$.

The first property in Definition 5.5 states that the initial state must be reachable. The second property says that if a configuration $c_{i-1} \in \{0,1\}^S$ is reachable after $i-1$ steps, then the state $c_i = \mathsf{Step}_i(c_{i-1}, 0)$ must also be reachable after i steps. This corresponds to the case of the evaluator taking a reachable configuration c_{i-1} from the first $i-1$ steps and reading the bit 0 on Step i. When $\beta_i = 1$, the second property says that the state $\mathsf{Step}_i(c_{i-1}, 1)$ must also be reachable after i steps. This corresponds to the evaluator taking c_{i-1} and reading the bit 1 on Step i. The final property says that if $\vec{\beta} \in \{0,1\}^K$ does not satisfy the policy, then the accepting states should *not* be reachable. Finally, Definition 5.5 allows the set $T_{\vec{\beta},i}$ to be a super-set of the actual set of reachable states that can arise from honest executions of P on inputs $\vec{\beta}' \in \{0,1\}^t$ that are consistent with $\vec{\beta} \in \{0,1\}^t$. The only requirement is that $T_{\vec{\beta},K}$ does *not* contain any accepting state (but could contain non-accepting states that are technically not reachable from an honest evaluation of P).

Remark 5.6 (CRS Size). In our unique witness map construction, the size of the common reference string grows with the maximum size of the description of $\text{Reachable}_{\vec{\beta},i}(c)$ rather than the cardinality of the set $T_{\vec{\beta},i}$. In settings where the description length of $T_{\vec{\beta},i}$ is much smaller than the size of the set itself (e.g., the case of weighted thresholds; see Remark 5.7), this yields constructions with a more compact CRS.

Remark 5.7 (Reachable States for Weighted Threshold Policies). A simple example of a monotone policy that can be computed by a read-once bounded-space Turing machine is a weighted threshold policy. A weighted threshold policy is parameterized by a set of (non-negative) weights $w_1, \ldots, w_K \in \mathbb{N}$ and a threshold $t \in \mathbb{N}$. An input $\vec{\beta} \in \{0,1\}^K$ satisfies the policy if $\sum_{i \in [K]} \beta_i w_i > t$. Suppose the maximum weight is W. It is easy to implement a weighted threshold policy with a read-once Turing machine with $S = \log(KW)$ bits of space. The state $c \in \{0,1\}^S$ is an accumulator that stores the current (weighted) sum and the $\text{Step}_i(c_{i-1}, \beta_i)$ circuit updates the accumulated value from c_{i-1} to $c_{i-1} + \beta_i w_i$. The set of accepting states consists of all values greater than the threshold. Moreover, this Turing machine has a simple and admissible set of reachable states $\{T_{\vec{\beta},i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [K]}$ where $T_{\vec{\beta},i} = \left\{t \in \{0,1\}^S : t \le \sum_{j \le i} \beta_j w_j\right\}$. Here, $\sum_{j \le i} \beta_j w_j$ is the *maximum* possible value that can arise after reading the first i bits of the input. As such, *every* valid configuration will be a value in the set $T_{\vec{\beta},i}$. It is easy to see that this ensemble of sets satisfies all of the admissibility requirements from Definition 5.5. Moreover, we can check membership in the set $T_{\vec{\beta},i}$ with a Boolean circuit $\text{Reachable}_{\vec{\beta},i}$ of size $\text{poly}(\log(KW))$ by simply hard-coding the threshold $\sum_{j \le i} \beta_j w_j$ within $\text{Reachable}_{\vec{\beta},i}$.

Construction 5.8 (Succinct Unique Witness Map for Read-Once Bounded-Space Policies). Let λ be a security parameter and $\mathcal{P}$ be a family of read-once bounded-space monotone policies that can be computed by read-once space-S Turing machines (see Definition 5.5). Let $s = s(\lambda, S)$ be a bound on the size of the Boolean circuit that computes a step functions Step_i associated with policies in $\mathcal{P}$. Without loss of generality, we assume that all of the Turing machines Γ computing a policy in $\mathcal{P}$ share the same initial state c_{init} and same accepting state c_{acc}. This is without loss of generality since we can always relabel the states of the Turing machine and apply the same relabeling to the step functions. Our construction relies on the following ingredients:

- Let $i\mathcal{O}$ be an indistinguishability obfuscator for Boolean circuits.
- Let $\Pi_{\text{SSB}} = (\text{SSB.Setup}, \text{SSB.Hash}, \text{SSB.Verify})$ be a somewhere statistically binding hash function.
- Let $\Pi_{\text{PPRF}} = (\text{F.KeyGen}, \text{F.Eval}, \text{F.Puncture})$ be a puncturable PRF. For a key k and an input x, we will write $\text{F}(k, x)$ to denote $\text{F.Eval}(k, x)$.
- Let $\text{G} \colon \{0,1\}^\ell \to \{0,1\}^m$ be an injective PRG with seed length $\ell = \ell(\lambda)$ and output length $m = m(\lambda)$.

Let $\lambda_{\text{obf}} = \lambda_{\text{obf}}(\lambda, S)$ and $\lambda_{\text{PRF}} = \lambda_{\text{PRF}}(\lambda, S)$ be polynomials in the security parameter which we will set in the security analysis. We construct a succinct unique witness map for batch NP and policy family $\mathcal{P}$ as follows:

- $\text{Setup}(1^\lambda, C, K)$: On input the security parameter λ, a Boolean relation $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and a Boolean policy $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, the setup algorithm proceeds as follows:
  - Sample two hash keys $\text{hk}_{\text{step}} \leftarrow \text{SSB.Setup}(1^\lambda, 1^s, 1^1, K, \varnothing)$ and $\text{hk}_{\text{inst}} \leftarrow \text{SSB.Setup}(1^\lambda, 1^n, 1^1, K, \varnothing)$ for hashing the step functions and the instances, respectively.
  - Let s' be the output length of $\text{SSB.Hash}(\text{hk}_{\text{step}}, \cdot)$ and let n' be the output length of $\text{SSB.Hash}(\text{hk}_{\text{inst}}, \cdot)$. Sample a PRF key
  $$k \leftarrow \text{F.KeyGen}\left(1^{\lambda_{\text{PRF}}}, 1^{s'+n'+\lceil \log K \rceil + S}, 1^\ell\right).$$
  We will denote domain elements for the PRF by a tuple $(\text{h}_{\text{step}}, \text{h}_{\text{inst}}, i, c)$ where h_{step} is a hash of the step functions, h_{inst} is a hash of the instances, $i \in [K]$ is an index, and $c \in \{0,1\}^S$ is a configuration of the Turing machine computing the policy.
  - Define the program MapProg as follows:

Fixed values: Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, number of instances K, hash keys $\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}$, the initial configuration $c_{\mathsf{init}} \in \{0,1\}^S$, and a PRF key k

Input: an index $i \in [K]$, a configuration $c \in \{0,1\}^S$, a step function $\mathsf{Step}\colon \{0,1\}^S \times \{0,1\} \to \{0,1\}$, an instance $x \in \{0,1\}^n$, hash values $\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}$, openings $\pi_{\mathsf{step}}, \pi_{\mathsf{inst}}$, a witness $w \in \{0,1\}^h$, and a signature $\sigma \in \{0,1\}^t$

On input $(i, c, \mathsf{Step}, x, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step}}, \pi_{\mathsf{inst}}, w, \sigma)$:

1. If $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{step}}, \mathsf{h}_{\mathsf{step}}, i, \mathsf{Step}, \pi_{\mathsf{step}}) = 0$, output $\perp$.
2. If $i = 1$ and $c \neq c_{\mathsf{init}}$, output $\perp$.
3. If $i > 1$ and $\mathsf{G}(\sigma) \neq \mathsf{G}(\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)))$, output $\perp$.
4. Compute the next configuration $c_i \in \{0,1\}^S$ as follows:

$$c_i = \begin{cases} \mathsf{Step}(c, 1) & \text{if } \mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) = 1 \text{ and } C(x, w) = 1 \\ \mathsf{Step}(c, 0) & \text{otherwise.} \end{cases}$$

5. Output $(c_i, \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)))$.

Figure 7: The mapping program $\mathsf{MapProg}[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k]$.

Let $\mathsf{size}_{\mathsf{MapProg}}$ be the maximum size of the program $\mathsf{MapProg}[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k]$ and the corresponding programs appearing in the proof of Theorem 5.13. Compute the obfuscated program

$$\mathsf{ObfMap} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{MapProg}}}, \mathsf{MapProg}[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k]).$$

– Define the program VerProg as follows:

Fixed values: number of instances K, the accepting configuration c_{acc}, and a PRF key k

Input: hash values $\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}$, and a signature $\sigma \in \{0,1\}^t$

On input $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$:

1. If $\mathsf{G}(\sigma) = \mathsf{G}(\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, K, c_{\mathsf{acc}})))$, output 1.
2. Else, output 0.

Figure 8: The verification program $\mathsf{VerProg}[K, c_{\mathsf{acc}}, k]$.

Let $\mathsf{size}_{\mathsf{VerProg}}$ be the maximum size of the program $\mathsf{VerProg}[K, c_{\mathsf{acc}}, k]$ and the corresponding programs appearing in the proof of Theorem 5.13. Compute the obfuscated program

$$\mathsf{ObfVer} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{VerProg}}}, \mathsf{VerProg}[K, c_{\mathsf{acc}}, k]).$$

– Output $\mathsf{crs} = (\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{ObfMap}, \mathsf{ObfVer})$.

- $\mathsf{Map}(\mathsf{crs}, P, (x_1, \ldots, x_K), (w_1, \ldots, w_K))$: On input $\mathsf{crs} = (\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{ObfMap}, \mathsf{ObfVer})$, a policy $P \in \mathcal{P}$ computed by a Turing machine $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_K \in \{0,1\}^h$, the mapping algorithm proceeds as follows:

  - Compute hashes of the step functions and the instances:

$$(\mathsf{h}_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)).$$

  - Initialize $c_0 = c_{\mathsf{init}}$ and $\sigma_0 = \bot$. For each $i \in [K]$, compute

$$(c_i, \sigma_i) = \mathsf{ObfMap}(i, c_{i-1}, \mathsf{Step}_i, x_i, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step},i}, \pi_{\mathsf{inst},i}, w_i, \sigma_{i-1}).$$

  - Output σ_K.

- $\mathsf{Verify}(\mathsf{crs}, P, (x_1, \ldots, x_K), \sigma)$: On input $\mathsf{crs} = (\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{ObfMap}, \mathsf{ObfVer})$, a policy $P \in \mathcal{P}$ computed by a Turing machine $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$, instances $x_1, \ldots, x_K \in \{0,1\}^n$, and a signature σ, the verification algorithm proceeds as follows:

  - Compute hashes of the step functions and the instances:

$$(\mathsf{h}_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)).$$

  - Output $\mathsf{ObfVer}(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$.

Theorem 5.9 (Completeness). *If $i\mathcal{O}$ and Π_{SSB} are correct, then Construction 5.8 is complete.*

Proof. Take any $\lambda \in \mathbb{N}$, Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, Boolean policy $P \in \mathcal{P}$ computed by a Turing machine $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$, instances $\vec{x} = (x_1, \ldots, x_K \in \{0,1\}^n)$, and witnesses $\vec{w} = (w_1, \ldots, w_K \in \{0,1\}^h)$ where $P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1$. Let $c_0 = c_{\mathsf{init}}$ and $c_i = \mathsf{Step}_i(c_{i-1}, C(x_i, w_i))$ for all $i \in [K]$. Then, by Definition 5.4, we have that $c_K = c_{\mathsf{acc}}$. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C, K)$. Then,

$$\mathsf{crs} = (\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{ObfMap}, \mathsf{ObfVer}).$$

Next, by correctness of $i\mathcal{O}$ and Π_{SSB}, $\mathsf{Map}(\mathsf{crs}, P, \vec{x}, \vec{w})$ will output $\sigma = \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, K, c_{\mathsf{acc}}))$, where

$$(\mathsf{h}_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)),$$

and k is the puncturable PRF key sampled by Setup and used to construct ObfMap and ObfVer. In this case, $\mathsf{VerProg}[K, c_{\mathsf{acc}}, k](\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma) = 1$, in which case $\mathsf{Verify}(\mathsf{crs}, P, \vec{x}, \sigma) = 1$, as required. Completeness follows. □

Theorem 5.10 (Uniqueness). *If $i\mathcal{O}$ and Π_{SSB} are correct, then Construction 5.8 is unique.*

Proof. Take any $\lambda \in \mathbb{N}$, Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, Boolean policy $P \in \mathcal{P}$ computed by a Turing machine $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$, instances $\vec{x} = (x_1, \ldots, x_K \in \{0,1\}^n)$, and witnesses $\vec{w} = (w_1, \ldots, w_K \in \{0,1\}^h)$ and $\vec{w}' = (w'_1, \ldots, w'_K \in \{0,1\}^h)$ where

$$P(C(x_1, w_1), \ldots, C(x_K, w_K)) = P(C(x_1, w'_1), \ldots, C(x_K, w'_K)) = 1.$$

Next, let $c_0 = c'_0 = c_{\mathsf{init}}$, and for each $i \in [K]$, define $c_i = \mathsf{Step}_i(c_{i-1}, C(x_i, w_i))$ and $c'_i = \mathsf{Step}_i(c'_{i-1}, C(x_i, w'_i))$. By Definition 5.4, this means $c_K = c'_K = c_{\mathsf{acc}}$. By correctness of $i\mathcal{O}$ and Π_{SSB}, this means $\mathsf{Map}(\mathsf{crs}, P, \vec{x}, \vec{w})$ and $\mathsf{Map}(\mathsf{crs}, P, \vec{x}, \vec{w}')$ will both output

$$\sigma = \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, K, c_{\mathsf{acc}})),$$

where

$$(\mathsf{h}_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)),$$

and k is the puncturable PRF key sampled by Setup and used to construct ObfMap and ObfVer. In particular, this means that

$$\mathsf{Map}(\mathsf{crs}, P, \vec{x}, \vec{w}) = \mathsf{Map}(\mathsf{crs}, P, \vec{x}, \vec{w}')$$

and uniqueness holds. □

Remark 5.11 (Local Evaluation)**.** Similar to the case of local decryption for succinct witness encryption (Definition 3.2), we can also define a "local" mapping algorithm for a succinct unique witness map for batch languages. In this setting, one can first preprocess a batch of statements $(x_1, \ldots, x_K)$ into a collection of (short) hints $(\mathsf{ht}_1, \ldots, \mathsf{ht}_K)$ with the property that given any policy P, and any set $\{(i, x_i, w_i, \mathsf{ht}_i)\}_{i \in T}$, the user can compute the canonical witness on $(x_1, \ldots, x_K)$ whenever $P(\beta_1, \ldots, \beta_K) = 1$ and

$$\beta_i = \begin{cases} 1 & i \in T \wedge C(x_i, w_i) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

A unique witness map that supports this type of local evaluation property immediately implies a succinct witness encryption scheme for batch languages with local decryption (via the construction described in Remark 5.2). It is easy to see that Construction 5.8 supports this local evaluation property. Namely, given a batch of instances $(x_1, \ldots, x_K)$, the hint ht_i associated with the i^{th} instance would simply be $\mathsf{ht}_i = (i, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},i})$ where $(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$. To compute the canonical witness given $\{(i, x_i, w_i, \mathsf{ht}_i)\}_{i \in T}$, the evaluator can run the Map algorithm in Construction 5.8, and simply input $\pi_{\mathsf{inst}} = w = \perp$ on all indices $i \notin T$.

5.1 Security and Succinctness

In this section, we show that Construction 5.8 satisfies selective security (Theorem 5.13) and succinctness (Theorem 5.33).

Sub-exponential hardness. We now proceed to give the security analysis for Construction 5.8. Security will rely on sub-exponential hardness assumptions of the underlying primitives. To facilitate this, we will formulate some of our security assumptions using (t, ε)-notation. We say that a primitive is (t, ε)-secure if, for all adversaries $\mathcal{A}$ running in time at most $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, the adversary's advantage is bounded by $\varepsilon(\lambda)$. If we say a primitive is secure (*without* giving an explicit (t, ε) dependence, then we mean that it satisfies the usual notion of $(1, \mathsf{negl}(\lambda))$ security). We now give the main security theorem and proof.

Pebbling lemma. We will also rely on the following pebbling lemma from [Ben89, GPSZ17]:

Lemma 5.12 (Pebbling Lemma [Ben89, GPSZ17])**.** *Take any positive integer $n \in \mathbb{N}$. Let $\vec{\tau}_0 = 0^n$. Then, there exists $N = O(n^{\log_2 3})$ and a sequence of strings $\vec{\tau}_1, \ldots, \vec{\tau}_N$ with the following properties:*

- $\vec{\tau}_N = 0^{n-1} \| 1$.
- *For all $i \in [N]$, $\vec{\tau}_{i-1}$ and $\vec{\tau}_i$ differ on a single index $j \in [n]$. Moreover, either $j = 1$ or $\tau_{i-1,j-1} = 1 = \tau_{i,j-1}$. In other words, either $\vec{\tau}_{i-1}$ and $\vec{\tau}_i$ differ only on the first index $j = 1$ or they differ on an index j and both $\vec{\tau}_{i-1}$ and $\vec{\tau}_i$ are equal to 1 in the preceding index $j - 1$.*
- *For all $i \in [N]$, the Hamming weight (i.e., the number of non-zero entries) in $\vec{\tau}_i$ is at most $1 + \log n$.*

Moreover, there exists an efficient and explicit algorithm that takes as input 1^n and outputs $\vec{\tau}_1, \ldots, \vec{\tau}_N$.

Theorem 5.13 (Selective Soundness)**.** *Suppose the primitives in Construction 5.8 satisfy the following properties:*

- *Suppose Π_{SSB} is correct, satisfies index hiding, and is somewhere statistically binding.*
- *Suppose Π_{PPRF} satisfies punctured correctness and $\big(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}\big)$-punctured pseudorandomness for some constant $\varepsilon_{\mathsf{PRF}} \in (0,1)$. Moreover, let $\lambda_{\mathsf{PRF}} = (\lambda + S)^{1/\varepsilon_{\mathsf{PRF}}}$.*
- *Suppose $i\mathcal{O}$ is $\big(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}\big)$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$. Moreover, let $\lambda_{\mathsf{obf}} = (\lambda + S)^{1/\varepsilon_{\mathsf{obf}}}$.*
- *Suppose* G *is secure.*

Specifically, we only assume standard polynomial security for Π_{SSB} and G *and sub-exponential security for Π_{PPRF} and $i\mathcal{O}$. Then Construction 5.8 is selectively sound.*

Proof. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an efficient non-uniform adversary for the selective soundness game. In particular, on input the security parameter 1^λ, algorithm $\mathcal{A}_0$ outputs a tuple $(C, P, (x_1, \ldots, x_K))$ together with some state information $\mathsf{st}_{\mathcal{A}}$ (of polynomial size). Algorithm $\mathcal{A}_1$ takes as input the state $\mathsf{st}_{\mathcal{A}}$ and the common reference string crs. For each $i \in [K]$, define the bit β_i as follows:

$$\beta_i := \begin{cases} 1 & \exists w_i \in \{0,1\}^h : C(x_i, w_i) = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

In addition, let $\{\mathsf{Reachable}_{\vec{\beta},i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [0,K]}$ be the Boolean circuits that compute an admissible set of reachable states associated with the policy P (as defined in Definition 5.5). Our reduction algorithms will take $(C, P, \vec{x})$, where $\vec{x} = (x_1, \ldots, x_k)$, the bits $\vec{\beta} = (\beta_1, \ldots, \beta_K)$, the description of the circuits $\{\mathsf{Reachable}_{\vec{\beta},i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [0,K]}$, and $\mathsf{st}_{\mathcal{A}}$ as non-uniform advice. Let $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$ be the description of the read-once Turing machine that computes the policy P. We now define a sequence of hybrid experiments.

- $\mathsf{Hyb}_{\mathsf{init}}$: This is the selective soundness game with adversary $\mathcal{A}$:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}_0(1^\lambda)$ outputs $(C, P, (x_1, \ldots, x_K))$ and $\mathsf{st}_{\mathcal{A}}$. For each $i \in [K]$, define the bits β_i according to Eq. (5.1).
  - If $P(\vec{\beta}) = 1$, the challenger outputs 0. Otherwise, the challenger invokes $\sigma \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, \mathsf{crs})$ where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C, K)$.
  - The output of the experiment is $b' = \mathsf{Verify}(\mathsf{crs}, P, (x_1, \ldots, x_K), \sigma)$.
- Hyb_τ for a string $\tau \in \{0,1\}^K$: Same as $\mathsf{Hyb}_{\mathsf{init}}$ except the challenger computes

$$(\mathsf{h}^*_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}^*_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$$

  and defines the following modified program $\mathsf{MapProg}_1$:

Fixed values: Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, number of instances K, hash keys $\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}$, the initial configuration $c_{\mathsf{init}} \in \{0,1\}^S$, a puncturable PRF key k, hash values $\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}$, a string $\vec{\beta} \in \{0,1\}^K$, and a string $\tau \in \{0,1\}^K$

Input: an index $i \in [K]$, a configuration $c \in \{0,1\}^S$, a step function $\mathsf{Step}\colon \{0,1\}^S \times \{0,1\} \to \{0,1\}$, an instance $x \in \{0,1\}^n$, hash values $\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}$, openings $\pi_{\mathsf{step}}, \pi_{\mathsf{inst}}$, a witness $w \in \{0,1\}^h$, and a signature $\sigma \in \{0,1\}^t$

On input $(i, c, \mathsf{Step}, x, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step}}, \pi_{\mathsf{inst}}, w, \sigma)$:

1. If $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, $\tau_i = 1$, and $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$, output $\bot$.
2. If $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{step}}, \mathsf{h}_{\mathsf{step}}, i, \mathsf{Step}, \pi_{\mathsf{step}}) = 0$, output $\bot$.
3. If $i = 1$ and $c \neq c_{\mathsf{init}}$, output $\bot$.
4. If $i > 1$ and $\mathsf{G}(\sigma) \neq \mathsf{G}(\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)))$, output $\bot$.
5. Compute the next configuration $c_i \in \{0,1\}^S$ as follows:
$$c_i = \begin{cases} \mathsf{Step}(c, 1) & \text{if } \mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) = 1 \text{ and } C(x, w) = 1 \\ \mathsf{Step}(c, 0) & \text{otherwise.} \end{cases}$$
6. Output $(c_i, \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)))$.

Figure 9: The mapping program $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]$.

When preparing the common reference string, the challenger now computes

$$\mathsf{ObfMap} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{MapProg}}}, \mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]).$$

- $\mathsf{Hyb}_{\mathsf{end}}$: Same as Hyb_τ for $\tau = 0^{K-1} \| 1$, except the challenger computes and defines the following modified program $\mathsf{VerProg}_1$:

Fixed values: number of instances K, the accepting configuration c_{acc}, a puncturable PRF key k, and hash values $\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}$

Input: hash values $\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}$, and a signature $\sigma \in \{0,1\}^t$

On input $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$:

1. If $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, output 0.
2. If $\mathsf{G}(\sigma) = \mathsf{G}(\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, K, c_{\mathsf{acc}})))$, output 1. Otherwise, output 0.

Figure 10: The verification program $\mathsf{VerProg}_1[K, c_{\mathsf{acc}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}]$.

When preparing the common reference string, the challenger now computes

$$\mathsf{ObfMap} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{MapProg}}}, \mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau])$$

where $\tau = 0^{K-1}\|1$, and

$$\mathsf{ObfVer} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{VerProg}}}, \mathsf{VerProg}_1[K, c_{\mathsf{acc}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}]).$$

Jumping ahead, we will show the following properties:

Property 1 Suppose $\tau = 0^K$. Then the success probability of $\mathcal{A}$ in $\mathsf{Hyb}_{\mathsf{init}}$ is only negligibly more than in Hyb_τ.

Property 2 Suppose τ, τ' differ only on index 1 (i.e., $\tau_1 = 0$ and $\tau'_1 = 1$). Then the success probability of $\mathcal{A}$ is only negligibly more in Hyb_τ than in $\mathsf{Hyb}_{\tau'}$.

Property 3 Suppose $\tau, \tau' \in \{0,1\}^K$ differ at a single index $i^* > 1$, and moreover $\tau_{i^*-1} = \tau'_{i^*-1} = 1$. Then the success probability of $\mathcal{A}$ is only negligibly more in Hyb_τ than in $\mathsf{Hyb}_{\tau'}$.

Property 4 Suppose $\tau = 0^{K-1}\|1$. Then the success probability of $\mathcal{A}$ is only negligibly more in Hyb_τ than in $\mathsf{Hyb}_{\mathsf{end}}$.

Property 5 The success probability of $\mathcal{A}$ in $\mathsf{Hyb}_{\mathsf{end}}$ is zero.

The claim then follows by Lemma 5.12, which provides an efficiently computable sequence of strings $\tau_1, \ldots, \tau_N \in \{0,1\}^K$ such that if we consider the sequence of hybrids $\mathsf{Hyb}_{\mathsf{init}}, \mathsf{Hyb}_{\tau_1}, \ldots, \mathsf{Hyb}_{\tau_N}, \mathsf{Hyb}_{\mathsf{end}}$, the success probability of $\mathcal{A}$ in each hybrid is only negligibly more than in the following hybrid. Since $N = \mathsf{poly}(K) = \mathsf{poly}(\lambda)$, and the success probability of $\mathcal{A}$ in $\mathsf{Hyb}_{\mathsf{end}}$ is zero, we can conclude that the success probability of $\mathcal{A}$ in $\mathsf{Hyb}_{\mathsf{init}}$ (i.e., the selective soundness game) is also $\mathsf{negl}(\lambda)$. We now formally prove the above properties.

Lemma 5.14 (**Property 1**). *Suppose $i\mathcal{O}$ is secure and that $\lambda_{\mathsf{obf}} \geq \lambda$. Let $\tau = 0^K$. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $|\Pr[\mathsf{Hyb}_{\mathsf{init}}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_\tau(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. It suffices to argue that the following two programs compute identical functionality:

- $\mathsf{MapProg}[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k]$ in $\mathsf{Hyb}_{\mathsf{init}}$; and
- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]$ in Hyb_τ.

By definition, the only difference in these two programs is the following additional check in $\mathsf{MapProg}_1$:

If $\mathsf{h}_{\mathsf{step}} = \mathsf{h}^*_{\mathsf{step}}$, $\mathsf{h}_{\mathsf{inst}} = \mathsf{h}^*_{\mathsf{inst}}$, $\tau_i = 1$, *and* $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$*: output* $\perp$.

However, since $\tau = 0^K$, this condition never triggers and the two programs compute identical functionality. The claim now follows by $i\mathcal{O}$ security. □

Lemma 5.15 (**Property 2**). *Suppose $i\mathcal{O}$ is secure and that $\lambda_{\mathsf{obf}} \geq \lambda$. Take any $\tau, \tau' \in \{0,1\}^K$ where $\tau_1 = 0$, $\tau'_1 = 1$, and $\tau_i = \tau'_i$ for all $i > 1$. Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_\tau(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau'}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. It suffices to argue that the following two programs compute identical functionality:

- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]$; and
- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau']$.

By definition, the only difference in these two programs is the following additional check in $\mathsf{MapProg}_1$:

If $\mathsf{h}_{\mathsf{step}} = \mathsf{h}^*_{\mathsf{step}}$, $\mathsf{h}_{\mathsf{inst}} = \mathsf{h}^*_{\mathsf{inst}}$, $\tau_i = 1$, *and* $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$*: output* $\perp$.

Take any input $(i, c, \mathsf{Step}, x, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step}}, \pi_{\mathsf{inst}}, w, \sigma)$ to these programs. We argue that the two programs above have identical behavior:

- Suppose $i \neq 1$. Then the behavior of the two programs are identical by construction, since $\tau_i = \tau'_i$ for all $i > 1$.
- Suppose $i = 1$ and $c \neq c_{\mathsf{init}}$. Then both programs output $\perp$.

- Suppose $i = 1$ and $c = c_{\mathsf{init}}$. Since the Reachable function describes an admissible set of reachable states, we have that $\mathsf{Reachable}_{\vec{\beta},0}(c) = 1$. In this case, the two programs again behave identically.

Thus the two programs compute identical functionality. The claim now follows by $i\mathcal{O}$ security. □

Lemma 5.16 (**Property 3**). *Suppose the following conditions hold:*

- *Suppose Π_{SSB} is correct, satisfies index hiding, and is somewhere statistically binding.*
- *Suppose Π_{PPRF} satisfies punctured correctness and $\left(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}\right)$-punctured pseudorandomness for some constant $\varepsilon_{\mathsf{PRF}} \in (0,1)$. Moreover, suppose $\lambda_{\mathsf{PRF}} = (\lambda + S)^{1/\varepsilon_{\mathsf{PRF}}}$.*
- *Suppose $i\mathcal{O}$ is $\left(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}\right)$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$. Moreover, suppose $\lambda_{\mathsf{obf}} = (\lambda + S)^{1/\varepsilon_{\mathsf{obf}}}$.*
- *Suppose G is secure and $m \geq \ell + \lambda$.*

Take any $\tau, \tau' \in \{0,1\}^K$ that differ on a single index $i^* > 1$, where $\tau_{i^*-1} = \tau'_{i^*-1} = 1$, $\tau_{i^*} = 0$, and $\tau'_{i^*} = 1$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{\tau}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau'}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. In the following, we will assume that our reduction algorithms are additionally provided (τ, τ', i^*) as part of their non-uniform advice. In the following, we will interpret configurations $c \in \{0,1\}^S$ as the binary representation of an S-bit integer. Then, for an integer $c^* \in \mathbb{Z}$, we say $c < c^*$ if the integer associated with c is less than or equal to the value of c^*. We now define an intermediate sequence of hybrid experiments.

- $\mathsf{Hyb}_{\tau,\mathsf{init}}$: Same as Hyb_{τ}, except the challenger samples the hash keys $\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}$ to be binding on the special index i^*. Namely, the challenger in this experiment samples

$$\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}\left(1^{\lambda}, 1^{s}, 1^{1}, K, \{i^*\}\right)$$
$$\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}\left(1^{\lambda}, 1^{n}, 1^{1}, K, \{i^*\}\right).$$

- Hyb_{τ,c^*} for $c^* \in [0, 2^S]$: Same as $\mathsf{Hyb}_{\tau,\mathsf{init}}$, except the challenger samples $y^* \xleftarrow{\mathrm{R}} \{0,1\}^{\ell}$ and then sets $z^* = \mathsf{G}(y^*)$, and defines the following modified program $\mathsf{MapProg}_2$:

Fixed values: Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, number of instances K, hash keys $\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}$, the initial configuration $c_{\mathsf{init}} \in \{0,1\}^S$, a puncturable PRF key k, hash values $\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}$, a string $\vec{\beta} \in \{0,1\}^K$, a string $\tau \in \{0,1\}^K$, an index $i^* \in [K]$, a string $z^* \in \{0,1\}^m$, and a value $c^* \in [0, 2^S]$

Input: an index $i \in [K]$, a configuration $c \in \{0,1\}^S$, a step function $\mathsf{Step}\colon \{0,1\}^S \times \{0,1\} \to \{0,1\}$, an instance $x \in \{0,1\}^n$, hash values $\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}$, openings $\pi_{\mathsf{step}}, \pi_{\mathsf{inst}}$, a witness $w \in \{0,1\}^h$, and a signature $\sigma \in \{0,1\}^t$

On input $(i, c, \mathsf{Step}, x, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step}}, \pi_{\mathsf{inst}}, w, \sigma)$:

1. If $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, $\tau_i = 1$, and $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$, output $\perp$.
2. If $i = i^*$, $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, $c < c^*$, $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$, and
$$\mathsf{G}(\sigma \oplus \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c))) \neq z^*,$$
output $\perp$.
3. If $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{step}}, \mathsf{h}_{\mathsf{step}}, i, \mathsf{Step}, \pi_{\mathsf{step}}) = 0$, output $\perp$.
4. If $i = 1$ and $c \neq c_{\mathsf{init}}$, output $\perp$.
5. If $i > 1$ and $\mathsf{G}(\sigma) \neq \mathsf{G}(\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)))$, output $\perp$.
6. Compute the next configuration $c_i \in \{0,1\}^S$ as follows:
$$c_i = \begin{cases} \mathsf{Step}(c, 1) & \text{if } \mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) = 1 \text{ and } C(x, w) = 1 \\ \mathsf{Step}(c, 0) & \text{otherwise.} \end{cases}$$
7. Output $(c_i, \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)))$.

Figure 11: The mapping program $\mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*]$.

When preparing the common reference string, the challenger now computes

$$\mathsf{ObfMap} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{MapProg}}}, \mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*])$$

- $\mathsf{Hyb}_{\tau,\mathsf{end},0}$: Same as $\mathsf{Hyb}_{\tau,2^S}$, except the challenger samples $z^* \xleftarrow{\mathrm{R}} \{0,1\}^m$.
- $\mathsf{Hyb}_{\tau,\mathsf{end},1}$: Same as $\mathsf{Hyb}_{\tau,\mathsf{end},0}$, except the challenger samples $\mathsf{hk}_{\mathsf{step}}$ and $\mathsf{hk}_{\mathsf{inst}}$ normally. Namely, the challenger in this experiment samples
$$\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^s, 1^1, K, \varnothing)$$
$$\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^1, K, \varnothing).$$

Claim 5.17. *Suppose Π_{SSB} satisfies index hiding. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*
$$|\Pr[\mathsf{Hyb}_\tau(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,\mathsf{init}}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. We define an intermediate hybrid $\mathsf{Hyb}'_{\tau,\mathsf{init}}$ which is the same as Hyb_τ, except the challenger samples $\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^s, 1^1, K, \{i^*\})$. The challenger samples $\mathsf{hk}_{\mathsf{inst}}$ as in Hyb_τ. It is easy to see that Hyb_τ and $\mathsf{Hyb}'_{\tau,\mathsf{init}}$ are computationally indistinguishable assuming index hiding security of Π_{SSB}. Formally, suppose
$$|\Pr[\mathsf{Hyb}_\tau(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}'_{\tau,\mathsf{init}}(\mathcal{A}) = 1]| \geq \varepsilon$$

for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ for the index hiding game. As noted earlier, we assume the preprocessing algorithm $\mathcal{B}_0$ outputs an advice string of the form

$$\mathsf{st}_{\mathcal{B}} = (C, P, \vec{x}, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, \{\mathsf{Reachable}_{\vec{\beta},i}\}_{\vec{\beta}\in\{0,1\}^K, i\in[0,K]}, \tau, \tau', i^*).$$

The online algorithm $\mathcal{B}_1$ then works as follows:

1. On input the security parameter 1^λ and the advice string $(C, P, \vec{x}, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, \{\mathsf{Reachable}_{\vec{\beta},i}\}_{\vec{\beta}\in\{0,1\}^K, i\in[0,K]}, \tau, \tau', i^*)$, algorithm $\mathcal{B}_1$ outputs the input length 1^s, the bound 1^1, the number of blocks K, and the set $\{i^*\}$.
2. The challenger responds with a hash key $\mathsf{hk}_{\mathsf{step}}$. Algorithm $\mathcal{B}_1$ samples $\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^1, K, \varnothing)$.
3. Let $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$ be the description of the Turing machine that computes P. Algorithm $\mathcal{B}_1$ now computes
$$(\mathsf{h}^*_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}^*_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)).$$
4. Algorithm $\mathcal{B}_1$ now samples the remaining components of the CRS as in Hyb_τ:
   - $k \leftarrow \mathsf{F.KeyGen}(1^{\lambda_{\mathsf{PRF}}}, 1^{s'+n'+\lceil \log K\rceil+S}, 1^\ell)$.
   - $\mathsf{ObfMap} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{MapProg}}}, \mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau])$.
   - $\mathsf{ObfVer} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{VerProg}}}, \mathsf{VerProg}[K, c_{\mathsf{acc}}, k])$.

   Algorithm $\mathcal{B}_1$ sets $\mathsf{crs} = (\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{ObfMap}, \mathsf{ObfVer})$ and invokes $\mathcal{A}_1$ on input $(\mathsf{st}_{\mathcal{A}}, \mathsf{crs})$. Algorithm $\mathcal{A}_1$ outputs σ.
5. Algorithm $\mathcal{B}_1$ outputs $\mathsf{ObfVer}(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$.

We consider the two possible distributions for $\mathsf{hk}_{\mathsf{step}}$:

- If the challenger samples $\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^s, 1^1, K, \varnothing)$, then algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_τ.
- If the challenger samples $\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^s, 1^1, K, \{i^*\})$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}'_{\tau,\mathsf{init}}$.

We conclude that algorithm $\mathcal{B}$ breaks index hiding with the same advantage as $\mathcal{A}$, which proves the claim. By an analogous argument (applied to $\mathsf{hk}_{\mathsf{inst}}$), we conclude that $\mathsf{Hyb}'_{\tau,\mathsf{init}}$ and $\mathsf{Hyb}_{\tau,\mathsf{init}}$ are also computationally indistinguishable. Claim 5.17 now follows by a standard hybrid argument. □

Claim 5.18. *Suppose $i\mathcal{O}$ is secure and that $\lambda_{\mathsf{obf}} \ge \lambda$. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_{\tau,\mathsf{init}}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,0}](\mathcal{A}) = 1| = \mathsf{negl}(\lambda).$$

Proof. If suffices to argue that the following two programs compute identical functionality:

- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]$ in $\mathsf{Hyb}_{\tau,\mathsf{init}}$; and
- $\mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*]$ in $\mathsf{Hyb}_{\tau,0}$.

By definition, the only difference in these two programs is the following additional check in $\mathsf{MapProg}_2$:

> *If $i = i^*$, $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, $c < c^*$, $\mathsf{Reachable}_{\vec{\beta},i-1}(c) \ne 1$, and*
> $$\mathsf{G}(\sigma \oplus \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c))) \ne z^*,$$
> *output $\perp$.*

However, since $c \in \{0,1\}^K$ and $c^* = 0$ in $\mathsf{Hyb}_{\tau,0}$, the condition $c < c^*$ is never satisfied. Thus, the two programs in $\mathsf{Hyb}_{\tau,\mathsf{init}}$ and $\mathsf{Hyb}_{\tau,0}$ compute identical functionality and the claim now follows by $i\mathcal{O}$ security. □

Claim 5.19. *Suppose the following conditions hold:*

- *Suppose Π_{SSB} is correct and somewhere statistically binding.*
- *Suppose Π_{PPRF} satisfies punctured correctness and $\left(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}\right)$-punctured pseudorandomness for some constant $\varepsilon_{\mathsf{PRF}} \in (0,1)$. Moreover, suppose $\lambda_{\mathsf{PRF}} = (\lambda + S)^{1/\varepsilon_{\mathsf{PRF}}}$.*
- *Suppose $i\mathcal{O}$ is $\left(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}\right)$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$. Moreover, suppose $\lambda_{\mathsf{obf}} = (\lambda + S)^{1/\varepsilon_{\mathsf{obf}}}$.*
- *Suppose G is secure and $m \geq \ell + \lambda$.*

Then, for all $c^* \in [0, 2^S - 1]$ and all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{\tau,c^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,c^*+1}(\mathcal{A}) = 1]| \leq \frac{\Omega(1)}{2^{\lambda+S}}.$$

Proof. We consider two cases depending on the value of $\mathsf{Reachable}_{\vec{\beta},i^*-1}(c^*)$.

Case 1: $\mathsf{Reachable}_{\vec{\beta},i^*-1}(c^*) = 1$**.** Suppose $\mathsf{Reachable}_{\vec{\beta},i^*-1}(c^*) = 1$. In this case, the programs

- $\mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*]$ in Hyb_{τ,c^*}; and
- $\mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^* + 1]$ in $\mathsf{Hyb}_{\tau,c^*+1}$.

compute identical functionality. This is because the only difference between the two programs is $\mathsf{MapProg}_2$ in $\mathsf{Hyb}_{\tau,c^*+1}$ contains the following additional check:

> *if* $i = i^*$, $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, $c = c^*$, $\mathsf{Reachable}_{\vec{\beta},i^*-1}(c) \neq 1$, *and*
>
> $$\mathsf{G}(\sigma \oplus \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c))) \neq z^*,$$
>
> *output* $\perp$.

When $\mathsf{Reachable}_{\vec{\beta},i^*-1}(c^*) = 1$, this check is vacuous. As such we conclude that the $\mathsf{MapProg}_2$ programs in the two experiments compute identical functionality, so the claim follows by $i\mathcal{O}$ security.

Case 2: $\mathsf{Reachable}_{\vec{\beta},i^*-1}(c^*) = 0$**.** Suppose $\mathsf{Reachable}_{\vec{\beta},i^*-1}(c^*) = 0$. In this case, we define an additional sequence of hybrid experiments:

- $\mathsf{Hyb}_{\tau,c^*,1}$: Same as Hyb_{τ,c^*} except after sampling the PRF key k, the challenger punctures it at the point $(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^* - 1, c^*)$. Specifically, the challenger computes

  $$k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)} \leftarrow \mathsf{F.Puncture}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^* - 1, c^*))$$

  and

  $$r^* = y^* \oplus \mathsf{F}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^* - 1, c^*)).$$

  Then, it defines the following program $\mathsf{MapProg}_3$:

Fixed values: Boolean relation $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, number of instances K, hash keys $\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}$, the initial configuration $c_{\mathsf{init}} \in \{0,1\}^S$, a puncturable PRF key $k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}$, hash values $\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}$, a string $\vec{\beta} \in \{0,1\}^K$, a string $\tau \in \{0,1\}^K$, an index $i^* \in [K]$, a string $z^* \in \{0,1\}^m$, a value $c^* \in [0, 2^S]$, a PRG seed $y^* \in \{0,1\}^t$, and a string $r^* \in \{0,1\}^t$

Input: an index $i \in [K]$, a configuration $c \in \{0,1\}^S$, a step function $\mathsf{Step}\colon \{0,1\}^S \times \{0,1\} \to \{0,1\}$, an instance $x \in \{0,1\}^n$, hash values $\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}$, openings $\pi_{\mathsf{step}}, \pi_{\mathsf{inst}}$, a witness $w \in \{0,1\}^h$, and a signature $\sigma \in \{0,1\}^t$

On input $(i, c, \mathsf{Step}, x, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step}}, \pi_{\mathsf{inst}}, w, \sigma)$:

1. If $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, $\tau_i = 1$, and $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$, output $\perp$.
2. If $i = i^*$, $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, $c < c^*$, $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$, and
$$\mathsf{G}\big(\sigma \oplus \mathsf{F}\big(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)\big)\big) \neq z^*,$$
output $\perp$.
3. If $i = i^*$, $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, $c = c^*$, and $\sigma \oplus r^* \neq y^*$, output $\perp$.
4. If $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{step}}, \mathsf{h}_{\mathsf{step}}, i, \mathsf{Step}, \pi_{\mathsf{step}}) = 0$, output $\perp$.
5. If $i = 1$ and $c \neq c_{\mathsf{init}}$, output $\perp$.
6. If $(i, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, c) \neq (i^*, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, c^*)$, $i > 1$ and
$$\mathsf{G}(\sigma) \neq \mathsf{G}\big(\mathsf{F}\big(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)\big)\big),$$
output $\perp$.
7. Compute the next configuration $c_i \in \{0,1\}^S$ as follows:
$$c_i = \begin{cases} \mathsf{Step}(c, 1) & \text{if } \mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) = 1 \text{ and } C(x, w) = 1 \\ \mathsf{Step}(c, 0) & \text{otherwise.} \end{cases}$$
8. Output $\big(c_i, \mathsf{F}\big(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)\big)\big)$.

Figure 12: The mapping program $\mathsf{MapProg}_3[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*, y^*, r^*]$.

When preparing the common reference string, the challenger now computes

$$\mathsf{ObfMap} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{MapProg}}}, \mathsf{MapProg}_3[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*, y^*, r^*])$$

- $\mathsf{Hyb}_{\tau, c^*, 2}$: Same as $\mathsf{Hyb}_{\tau, c^*, 1}$ except the challenger now samples $r^* \xleftarrow{\text{R}} \{0,1\}^t$.
- $\mathsf{Hyb}_{\tau, c^*, 3}$: Same as $\mathsf{Hyb}_{\tau, c^*, 2}$ except the challenger now sets $r^* = \mathsf{F}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*))$.

Claim 5.20. *Suppose Π_{SSB} is somewhere statistically binding, Π_{PPRF} is correct, G is injective, and $i\mathcal{O}$ is $\big(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}\big)$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$. Suppose moreover that $\lambda_{\mathsf{obf}} = (\lambda + S)^{1/\varepsilon_{\mathsf{obf}}}$. Then, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

$$|\Pr[\mathsf{Hyb}_{\tau, c^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau, c^*, 1}(\mathcal{A}) = 1]| \leq \frac{1}{2^{\lambda + S}}.$$

Proof. We start by arguing that the following two programs compute identical functionality:

- $\mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_K, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*]$
- $\mathsf{MapProg}_3[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*, y^*, r^*]$

Take any input $(i, c, \mathsf{Step}, x, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step}}, \pi_{\mathsf{inst}}, w, \sigma)$ to these programs. First, if $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) \neq (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$, then by punctured correctness, this means

$$\mathsf{F}\big(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \cdot, \cdot)\big) = \mathsf{F}\big(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \cdot, \cdot)\big).$$

In this case, the behavior of the two programs is identical. Thus, for the remainder of the analysis, it suffices to consider the case where $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$. We now consider the programs' behavior depending on the value of i:

- Suppose $i < i^* - 1$ or $i > i^*$. This means $i - 1 \neq i^* - 1$. By punctured correctness, this means

  $$\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)) = \mathsf{F}(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)).$$

  Hence, the check in Step 6 of $\mathsf{MapProg}_3$ coincides with the corresponding check in $\mathsf{MapProg}_2$. Since we also have $i \neq i^* - 1$, punctured correctness also implies that for all $c_i \in \{0,1\}^S$,

  $$\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)) = \mathsf{F}(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)).$$

  This means the output of Step 8 of $\mathsf{MapProg}_3$ coincides with the output in $\mathsf{MapProg}_2$. We conclude that the two programs behave identically on all inputs where $i < i^* - 1$.

- Suppose $i = i^* - 1$. Then, $i - 1 \neq i^* - 1$. By the same argument as the previous case, the only possible difference in the behavior of the two programs is the computation of
  - $\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i))$ in Step 7 of $\mathsf{MapProg}_2$; and the computation of
  - $\mathsf{F}\big(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)\big)$ in Step 8 of $\mathsf{MapProg}_3$.

  First, by assumption $\tau_{i^*-1} = 1$, so $\tau_i = 1$ in this case. We now consider the possibilities depending on the value of $c \in \{0,1\}^S$:

  - Suppose $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$. Since $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$ and $\tau_i = 1$, both programs output $\perp$ in this case.
  - Suppose $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) = 1$. First, recall that $\mathsf{hk}_{\mathsf{step}}$ and $\mathsf{hk}_{\mathsf{inst}}$ are somewhere statistically binding at index i. Next, $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$ where

    $$(\mathsf{h}^*_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
    $$(\mathsf{h}^*_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$$

    Thus, with overwhelming probability over the choice of $\mathsf{hk}_{\mathsf{step}}$ and $\mathsf{hk}_{\mathsf{inst}}$:
    * If $\mathsf{Step} \neq \mathsf{Step}_i$, then $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{step}}, \mathsf{h}_{\mathsf{step}}, i, \mathsf{Step}, \pi_{\mathsf{step}}) = 0$.
    * If $x \neq x_i$, then $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) = 0$.

    Thus, if $\mathsf{Step} \neq \mathsf{Step}_i$, with overwhelming probability over the choice of $\mathsf{hk}_{\mathsf{step}}$, both programs output $\perp$. It suffices to analyze inputs where $\mathsf{Step} = \mathsf{Step}_i$. We now consider two possibilities:
    * Suppose $C(x, w) = 0$ or $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) = 0$. Then, both programs compute the configuration

      $$c_i = \mathsf{Step}(c, 0) = \mathsf{Step}_i(c, 0).$$

      By assumption, $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) = 1$. Since Reachable computes an admissible set of reachable states (Definition 5.5), this means $\mathsf{Reachable}_{\vec{\beta}, i}(c_i) = 1$. Correspondingly, this means $c_i \neq c^*$ (since we are working with the case where $\mathsf{Reachable}_{\vec{\beta}, i^*-1}(c^*) = 0$). By punctured correctness, this means

      $$\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)) = \mathsf{F}\big(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)\big)$$

      and the programs behave identically.

    * Conversely, suppose $C(x, w) = 1$ and $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) = 1$. As argued above, with overwhelming probability over the choice of $\mathsf{hk}_{\mathsf{inst}}$, this means $x = x_i$. This means $C(x_i, w) = 1$ so by definition of β_i (see Eq. (5.1)), this means that $\beta_i = 1$. Let $c_i = \mathsf{Step}(c, 1) = \mathsf{Step}_i(c, 1)$ be the configuration computed by both programs. Since $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) = 1$, $\beta_i = 1$, and $\mathsf{Reachable}$ computes an admissible set of reachable states, this means that $\mathsf{Reachable}_{\vec{\beta}, i}(c_i) = 1$. As in the previous case, this means $c_i \neq c^*$ (since $\mathsf{Reachable}_{\vec{\beta}, i^*-1}(c^*) = 0$). By punctured correctness, we have that $\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)) = \mathsf{F}(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i))$.

  We conclude that the output of both programs are identical.

- Suppose $i = i^*$. We consider the possibilities depending on the value of c:
  - Suppose $c \neq c^*$. By punctured correctness,

$$\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)) = \mathsf{F}\big(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)\big)$$

    and the two programs implement identical checks.
  - Suppose $c = c^*$. Then, $\mathsf{MapProg}_2$ checks that $\mathsf{G}(\sigma) = \mathsf{G}(\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)))$ whereas $\mathsf{MapProg}_3$ checks that $\sigma \oplus r^* = y^*$. If the condition is not satisfied, then the respective programs output $\perp$. Since G is injective, the check in $\mathsf{MapProg}_2$ is equivalent to checking

$$\sigma = \mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)). \tag{5.2}$$

    In $\mathsf{Hyb}_{\tau, c^*, 1}$, the challenger sets

$$r^* = \mathsf{F}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)) \oplus y^*,$$

    so the condition $\sigma \oplus r^* = y^*$ is precisely equivalent to Eq. (5.2) given that $(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*) = (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c)$.

  Finally, since $i \neq i^* - 1$, for all $c_i \in \{0,1\}^S$, punctured correctness implies that

$$\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)) = \mathsf{F}(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i)).$$

  Hence the behavior of the two programs is identical in the two experiments.

Thus the two programs compute identical functionality. The claim now follows by sub-exponential iO security. Formally, suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$, we have that

$$|\Pr[\mathsf{Hyb}_{\tau, c^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau, c^*, 1}(\mathcal{A}) = 1]| > \frac{1}{2^{\lambda + S}}.$$

Let $\Lambda_{\mathcal{B}} = \{(\lambda + S)^{1/\varepsilon_{\mathsf{obf}}} : \lambda \in \Lambda_{\mathcal{A}}\}$. Since S is non-negative and $\Lambda_{\mathcal{A}}$ is infinite, the set $\Lambda_{\mathcal{B}}$ is also infinite. We now use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct a non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ such that for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, the advantage of $\mathcal{B}$ is at least $1/2^{\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$. The preprocessing algorithm $\mathcal{B}_0$ proceeds as follows:

1. On input $1^{\lambda_{\mathsf{obf}}}$, algorithm $\mathcal{B}_0$ first checks if there exists $\lambda \in \Lambda_{\mathcal{A}}$ such that $\lambda_{\mathsf{obf}} = (\lambda + S)^{1/\varepsilon_{\mathsf{obf}}}$. If no such λ exists, then algorithm $\mathcal{B}_0$ outputs $\perp$. Otherwise, it sets λ to be the smallest such value that satisfies the condition.

2. If there exists $\lambda \in \Lambda_{\mathcal{A}}$ satisfying the above condition, then algorithm $\mathcal{B}$ runs $\mathcal{A}_0$ on input 1^λ to obtain $(C, P, \vec{x}, \mathsf{st}_{\mathcal{A}})$. It then computes $\vec{\beta} \in \{0,1\}^K$ according to Eq. (5.1) and the Boolean circuits $\{\mathsf{Reachable}_{\vec{\beta}, i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [0,K]}$ associated with P.

3. Algorithm $\mathcal{B}_0$ outputs the advice string

$$\mathsf{st}_{\mathcal{B}} = (C, P, \vec{x}, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, \{\mathsf{Reachable}_{\vec{\beta}, i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [0,K]}, \tau, \tau', i^*, c^*, \lambda). \tag{5.3}$$

The online algorithm $\mathcal{B}_1$ then proceeds as follows:

1. On input the security parameter $1^{\lambda_{\mathsf{obf}}}$ and the advice string $\mathsf{st}_{\mathcal{B}}$ (parsed according to Eq. (5.3)), algorithm $\mathcal{B}_1$ samples
$$\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^s, 1^1, K, \{i^*\})$$
$$\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^1, K, \{i^*\}).$$
2. Let $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$ be the description of the Turing machine that computes P. Algorithm $\mathcal{B}_1$ computes
$$(\mathsf{h}^*_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}^*_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)).$$
3. Next, algorithm $\mathcal{B}_1$ samples a PRF key $k \leftarrow \mathsf{F.KeyGen}(1^{\lambda_{\mathsf{PRF}}}, 1^{s'+n'+\lceil \log K \rceil + S}, 1^\ell)$. It also computes the punctured key
$$k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)} \leftarrow \mathsf{F.Puncture}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)).$$
4. Algorithm $\mathcal{B}_1$ samples $y^* \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ and sets $z^* = \mathsf{G}(y^*)$. Finally, it computes
$$r^* = y^* \oplus \mathsf{F}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)).$$
5. Algorithm $\mathcal{B}_1$ gives the programs
$$\mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*]$$
and
$$\mathsf{MapProg}_3[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*, y^*, r^*]$$
to the $i\mathcal{O}$ challenger. The $i\mathcal{O}$ challenger responds with an obfuscated program ObfMap.
6. Finally, algorithm $\mathcal{B}_1$ computes $\mathsf{ObfVer} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{VerProg}}}, \mathsf{VerProg}[K, c_{\mathsf{acc}}, k])$. It defines the common reference string $\mathsf{crs} = (\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{ObfMap}, \mathsf{ObfVer})$ and invokes $\mathcal{A}_1$ on input $(\mathsf{st}_{\mathcal{A}}, \mathsf{crs})$. Algorithm $\mathcal{A}_1$ outputs σ.
7. Algorithm $\mathcal{B}_1$ outputs $\mathsf{ObfVer}(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$.

By construction, if ObfMap is an obfuscation of $\mathsf{MapProg}_2$, then algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_{τ,c^*}, whereas if ObfMap is an obfuscation of $\mathsf{MapProg}_3$, then algorithm $\mathcal{B}_1$ perfectly simulates an execution of $\mathsf{Hyb}_{\tau,c^*,1}$. We claim that for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$ algorithm $\mathcal{B}$ breaks security of $i\mathcal{O}$ with advantage $2^{-(\lambda+S)} = 2^{-\lambda_{\mathsf{obf}}^{1/\varepsilon_{\mathsf{obf}}}}$, which contradicts sub-exponential security of $i\mathcal{O}$. ☐

Claim 5.21. *Suppose* Π_{PPRF} *satisfies* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-punctured pseudorandomness for some constant* $\varepsilon_{\mathsf{PRF}} \in (0,1)$. *Suppose moreover that* $\lambda_{\mathsf{PRF}} = (\lambda + S)^{1/\varepsilon_{\mathsf{PRF}}}$. *Then, there exists* $\lambda_{\mathcal{A}} \in \mathbb{N}$ *such that for all* $\lambda \geq \lambda_{\mathcal{A}}$,
$$|\Pr[\mathsf{Hyb}_{\tau,c^*,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,c^*,2}(\mathcal{A}) = 1]| \leq \frac{1}{2^{\lambda+S}}.$$

Proof. Suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$, we have that
$$|\Pr[\mathsf{Hyb}_{\tau,c^*,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,c^*,2}(\mathcal{A}) = 1]| > \frac{1}{2^{\lambda+S}}.$$

Let $\Lambda_{\mathcal{B}} = \{(\lambda + S)^{1/\varepsilon_{\mathsf{PRF}}} : \lambda \in \Lambda_{\mathcal{A}}\}$. Since S is non-negative and $\Lambda_{\mathcal{A}}$ is infinite, the set $\Lambda_{\mathcal{B}}$ is also infinite. We now use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct a non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ such that for all $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, the advantage of $\mathcal{B}$ is at least $1/2^{\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}$. The preprocessing algorithm $\mathcal{B}_0$ proceeds as follows (this is entirely analogous to the preprocessing algorithm from the proof of Claim 5.20):

1. On input $1^{\lambda_{\mathsf{PRF}}}$, algorithm $\mathcal{B}_0$ checks if there exists $\lambda \in \Lambda_{\mathcal{A}}$ such that $\lambda_{\mathsf{PRF}} = (\lambda + S)^{1/\varepsilon_{\mathsf{PRF}}}$. If no such λ exists, then algorithm $\mathcal{B}_0$ outputs $\perp$. Otherwise, it sets λ to be the smallest such value that satisfies the condition.
2. If there exists $\lambda \in \Lambda_{\mathcal{A}}$ satisfying the above condition, then algorithm $\mathcal{B}$ runs $\mathcal{A}_0$ on input 1^λ to obtain $(C, P, \vec{x}, \mathsf{st}_{\mathcal{A}})$. It then computes $\vec{\beta} \in \{0,1\}^K$ according to Eq. (5.1) and the Boolean circuits $\{\mathsf{Reachable}_{\vec{\beta},i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [0,K]}$ associated with P.
3. Algorithm $\mathcal{B}_0$ outputs the advice string

$$\mathsf{st}_{\mathcal{B}} = (C, P, \vec{x}, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, \{\mathsf{Reachable}_{\vec{\beta},i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [0,K]}, \tau, \tau', i^*, c^*, \lambda). \tag{5.4}$$

The online algorithm $\mathcal{B}_1$ then proceeds as follows:

1. On input the security parameter $1^{\lambda_{\mathsf{PRF}}}$ and the advice string $\mathsf{st}_{\mathcal{B}}$ (parsed according to Eq. (5.4)), algorithm $\mathcal{B}_1$ samples

$$\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^s, 1^1, K, \{i^*\})$$
$$\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^1, K, \{i^*\}).$$

2. Let $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$ be the description of the Turing machine that computes P. Algorithm $\mathcal{B}_1$ computes

$$(\mathsf{h}^*_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}^*_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)).$$

3. Next, algorithm $\mathcal{B}_1$ outputs the input length $1^{s'+n'+\lceil \log K \rceil + S}$, the output length 1^ℓ, and the challenger point $(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)$ to the challenger. The challenger replies with a punctured key $k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}$ and a challenge $\zeta \in \{0,1\}^\ell$.
4. Algorithm $\mathcal{B}_1$ samples $y^* \xleftarrow{\text{R}} \{0,1\}^\lambda$ and sets $z^* = \mathsf{G}(y^*)$. Finally, it computes $r^* = y^* \oplus \zeta$.
5. Algorithm $\mathcal{B}_1$ then computes

$$\mathsf{ObfMap} \leftarrow i\mathcal{O}(\mathsf{MapProg}_3[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, c^*, y^*, r^*])$$

and $\mathsf{ObfVer} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{VerProg}}}, \mathsf{VerProg}[K, c_{\mathsf{acc}}, k])$. It sets $\mathsf{crs} = (\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{ObfMap}, \mathsf{ObfVer})$ and invokes $\mathcal{A}_1$ on input $(\mathsf{st}_{\mathcal{A}}, \mathsf{crs})$. Algorithm $\mathcal{A}_1$ outputs σ.
6. Algorithm $\mathcal{B}_1$ outputs $\mathsf{ObfVer}(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$.

By construction, the challenger samples $k \leftarrow \mathsf{F.KeyGen}(1^{\lambda_{\mathsf{PRF}}}, 1^{s'+n'+\lceil \log K \rceil + S}, 1^\ell)$ and

$$k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)} \leftarrow \mathsf{F.Puncture}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*)).$$

We now consider the two possibilities for the challenge value ζ:

- Suppose $\zeta = \mathsf{F}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, i^*-1, c^*))$. In this case, $r^* = y^* \oplus \zeta$ is distributed according to the specification of $\mathsf{Hyb}_{\tau,c^*,1}$.
- Suppose $\zeta \xleftarrow{\text{R}} \{0,1\}^\ell$. In particular, ζ in this case is independent of all other components in the experiment. This means the distribution of $r^* = y^* \oplus \zeta$ is also uniform over $\{0,1\}^\ell$, which coincides with the distribution of r^* in $\mathsf{Hyb}_{\tau,c^*,2}$.

We conclude that for all $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$ algorithm $\mathcal{B}$ breaks punctured pseudorandomness with advantage $2^{-(\lambda+S)} = 2^{-\lambda_{\mathsf{PRF}}^{1/\varepsilon_{\mathsf{PRF}}}}$, which contradicts sub-exponential security of Π_{PPRF}. ☐

Claim 5.22. *Suppose* Π_{PPRF} *satisfies* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-punctured pseudorandomness for some constant* $\varepsilon_{\mathsf{PRF}} \in (0,1)$*. Suppose moreover that* $\lambda_{\mathsf{PRF}} = (\lambda + S)^{1/\varepsilon_{\mathsf{PRF}}}$*. Then, there exists* $\lambda_{\mathcal{A}} \in \mathbb{N}$ *such that for all* $\lambda \geq \lambda_{\mathcal{A}}$*,*

$$|\Pr[\mathsf{Hyb}_{\tau,c^*,2}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,c^*,3}(\mathcal{A}) = 1]| \leq \frac{1}{2^{\lambda+S}}.$$

Proof. Follows by a similar argument as in the proof of Claim 5.21. The only difference is the reduction algorithm sets $r^* = \zeta$ (rather than $r^* = y^* \oplus \zeta$). □

Claim 5.23. *Suppose* Π_{PPRF} *is correct,* G *is injective, and* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$*-secure for some constant* $\varepsilon_{\mathsf{obf}} \in (0,1)$*. Suppose moreover that* $\lambda_{\mathsf{obf}} = (\lambda + S)^{1/\varepsilon_{\mathsf{obf}}}$*. Then, there exists* $\lambda_{\mathcal{A}} \in \mathbb{N}$ *such that for all* $\lambda \geq \lambda_{\mathcal{A}}$*,*

$$|\Pr[\mathsf{Hyb}_{\tau,c^*,3}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,c^*+1}(\mathcal{A}) = 1]| \leq \frac{1}{2^{\lambda+S}}.$$

Proof. Follows by a similar argument as in the proof of Claim 5.20. □

Claim 5.19 now follows by combining Claims 5.20 to 5.23. □

Claim 5.24. *Suppose* G *is pseudorandom. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$|\Pr[\mathsf{Hyb}_{\tau,2^S}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,\mathsf{end},0}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Formally, suppose

$$|\Pr[\mathsf{Hyb}_{\tau,2^S}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,\mathsf{end},0}(\mathcal{A}) = 1]| \geq \varepsilon$$

for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ that breaks security of G. As noted earlier, we assume the preprocessing algorithm $\mathcal{B}_0$ outputs an advice string of the form

$$\mathsf{st}_{\mathcal{B}} = (C, P, \vec{x}, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, \{\mathsf{Reachable}_{\vec{\beta},i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [0,K]}, \tau, \tau', i^*).$$

The online algorithm $\mathcal{B}_1$ then works as follows:

1. On input the security parameter 1^λ, the advice string $(C, P, \vec{x}, \mathsf{st}_{\mathcal{A}}, \vec{\beta}, \{\mathsf{Reachable}_{\vec{\beta},i}\}_{\vec{\beta} \in \{0,1\}^K, i \in [0,K]}, \tau, \tau', i^*)$, and the challenge $z^* \in \{0,1\}^m$, algorithm $\mathcal{B}_1$ samples

$$\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^s, 1^1, K, \{i^*\})$$
$$\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^n, 1^1, K, \{i^*\}).$$

2. Let $\Gamma = (\mathsf{Step}_1, \ldots, \mathsf{Step}_K, c_{\mathsf{init}}, c_{\mathsf{acc}})$ be the description of the Turing machine that computes P. Algorithm $\mathcal{B}_1$ now computes

$$(\mathsf{h}^*_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}^*_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)).$$

   It also computes

   - $k \leftarrow \mathsf{F.KeyGen}(1^{\lambda_{\mathsf{PRF}}}, 1^{s'+n'+\lceil \log K \rceil + S}, 1^\ell)$.
   - $\mathsf{ObfMap} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{MapProg}}}, \mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, 2^S])$.
   - $\mathsf{ObfVer} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{VerProg}}}, \mathsf{VerProg}[K, c_{\mathsf{acc}}, k])$.

   Algorithm $\mathcal{B}_1$ sets $\mathsf{crs} = (\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, \mathsf{ObfMap}, \mathsf{ObfVer})$ and invokes $\mathcal{A}_1$ on input $(\mathsf{st}_{\mathcal{A}}, \mathsf{crs})$. Algorithm $\mathcal{A}_1$ outputs σ.

3. Algorithm $\mathcal{B}_1$ outputs $\mathsf{ObfVer}(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$.

We consider the two possible distributions for z^*:

- If $z^* = \mathsf{G}(y^*)$ where $y^* \xleftarrow{\mathrm{R}} \{0,1\}^\ell$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{\tau,2^S}$.
- If $z^* \xleftarrow{\mathrm{R}} \{0,1\}^m$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{\tau,\mathsf{end},0}$.

We conclude that algorithm $\mathcal{B}$ breaks pseudorandomness of G with the same advantage as $\mathcal{A}$, which proves the claim. □

Claim 5.25. *Suppose* Π_{SSB} *satisfies index hiding. Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{\tau,\mathsf{end},0}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,\mathsf{end},1}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Follows by the same argument as the proof of Claim 5.17. □

Claim 5.26. *Suppose iO is secure and that* $\lambda_{\mathsf{obf}} \geq \lambda$. *Suppose also that* $m \geq \ell + \lambda$. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{\tau,\mathsf{end},1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau'}(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. It suffices to argue that the following two programs compute identical functionality:

- $\mathsf{MapProg}_2[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau, i^*, z^*, 2^S]$ in $\mathsf{Hyb}_{\tau,\mathsf{end},1}$; and
- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau']$.

Take any input $(i, c, \mathsf{Step}, x, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step}}, \pi_{\mathsf{inst}}, w, \sigma)$ to these two programs. We consider the following possibilities:

- Suppose $i \neq i^*$. By assumption, this means $\tau_i = \tau'_i$, so the two programs behave identically.
- Suppose $i = i^*$. This means that $\tau_i = 0$ and $\tau'_i = 1$. We consider two possibilities:
  - Suppose $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$ and $\mathsf{Reachable}_{\vec{\beta},i-1}(c) \neq 1$. In this case, $\mathsf{MapProg}_1$ always outputs $\perp$. Consider the behavior in $\mathsf{MapProg}_2$. First, the condition $c < 2^S$ always holds. Moreover, in $\mathsf{Hyb}_{\tau,\mathsf{end},1}$, the challenger samples $z^* \xleftarrow{\mathrm{R}} \{0,1\}^m$. Since $m \geq \ell + \lambda$, with overwhelming probability over the choice of z^*, we have that z^* is not in the image of G. In this case, the check in Step 2 of $\mathsf{MapProg}_2$ always triggers and the program outputs $\perp$.
  - Suppose $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) \neq (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$ or $\mathsf{Reachable}_{\vec{\beta},i-1}(c) = 1$. In this case, the behavior of the two programs are identical by definition.

We conclude that the two programs compute identical functionality. The claim now follows by *iO* security. □

Completing the proof of Lemma 5.16. We now return to the proof of Lemma 5.16. By Claim 5.19, for all $c^* \in [0, 2^S - 1]$ and all sufficiently-large $\lambda \in \mathbb{N}$, we have that

$$|\Pr[\mathsf{Hyb}_{\tau,c^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,c^*+1}(\mathcal{A}) = 1]| \leq \frac{\Omega(1)}{2^{\lambda+S}}.$$

By a hybrid argument, this means

$$|\Pr[\mathsf{Hyb}_{\tau,0}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,2^S}(\mathcal{A}) = 1]| \leq \frac{\Omega(1)}{2^{\lambda}}.$$

Combined with Claims 5.17, 5.18 and 5.24 to 5.26, we conclude via a hybrid argument that

$$|\Pr[\mathsf{Hyb}_{\tau}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau'}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$ □

Lemma 5.27 (**Property 4**). *Let* $\tau = 0^{K-1}\|1$. *Then for all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_\tau(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{end}}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. We introduce an intermediate sequence of hybrids.

- $\mathsf{Hyb}_{\tau,1}$: Same as Hyb_τ except the challenger samples the hash keys $\mathsf{hk}_{\mathsf{step}}$ and $\mathsf{hk}_{\mathsf{inst}}$ to be binding on index K. Namely, the challenger samples

$$\mathsf{hk}_{\mathsf{step}} \leftarrow \mathsf{SSB.Setup}(1^{\lambda_{\mathsf{SSB}}}, 1^s, 1^1, K, \{K\})$$
$$\mathsf{hk}_{\mathsf{inst}} \leftarrow \mathsf{SSB.Setup}(1^{\lambda_{\mathsf{SSB}}}, 1^n, 1^1, K, \{K\}).$$

- $\mathsf{Hyb}_{\tau,2}$: Same as $\mathsf{Hyb}_{\tau,1}$ except the challenger punctures the PRF key k at $(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})$. Namely, the challenger computes

$$k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})} \leftarrow \mathsf{F.Puncture}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})).$$

  The challenger also computes $z^* = \mathsf{F}\big(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})\big)$ and defines the following program $\mathsf{VerProg}_2$:

Fixed values: number of instances K, the accepting configuration c_{acc}, a puncturable PRF key $k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}$, hash values $\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}$, and a string $z^* \in \{0,1\}^m$

Input: hash values $\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}$, and a signature $\sigma \in \{0,1\}^t$

On input $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$:

1. If $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$ and $\mathsf{G}(\sigma) = z^*$, output 1.
2. If $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) \neq (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$ and $\mathsf{G}(\sigma) = \mathsf{G}(\mathsf{F}(k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, K, c_{\mathsf{acc}})))$, output 1.
3. Otherwise, output 0.

Figure 13: The verification program $\mathsf{VerProg}_2[K, c_{\mathsf{acc}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, z^*]$.

  When preparing the common reference string, the challenger now computes

$$\mathsf{ObfMap} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{MapProg}}}, \mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]).$$

  and

$$\mathsf{ObfVer} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{\mathsf{size}_{\mathsf{VerProg}}}, \mathsf{VerProg}_2[K, c_{\mathsf{acc}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, z^*]).$$

- $\mathsf{Hyb}_{\tau,3}$: Same as $\mathsf{Hyb}_{\tau,3}$ except the challenger samples $z^* \xleftarrow{\mathrm{R}} \{0,1\}^m$.

Claim 5.28. *Suppose* Π_{SSB} *satisfies index hiding. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_\tau(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,1}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Follows by the same argument as the proof of Claim 5.17. □

Claim 5.29. *Suppose* Π_{PPRF} *is correct,* $i\mathcal{O}$ *is secure, and* $\lambda_{\mathsf{obf}} \geq \lambda$. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{\tau,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,2}(\mathcal{A})]| \leq \mathsf{negl}(\lambda).$$

Proof. First we note that the following two programs compute identical functionality:

- $\mathsf{VerProg}[K, c_{\mathsf{acc}}, k]$ in $\mathsf{Hyb}_{\tau,1}$; and
- $\mathsf{VerProg}_2[K, c_{\mathsf{acc}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, z^*]$ in $\mathsf{Hyb}_{\tau,2}$.

Consider any input $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$ to these programs.

- Suppose $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$. Then both programs output 1 if and only if
$$\mathsf{G}(\sigma) = z^* = \mathsf{G}(\mathsf{F}(k, (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}}))).$$
- Suppose $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) \neq (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$. Then, by punctured correctness, both programs output 1 if and only if
$$\mathsf{G}(\sigma) = \mathsf{G}(\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, K, c_{\mathsf{acc}}))).$$

Next, we show that the following two programs also compute identical functionality:

- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]$ in $\mathsf{Hyb}_{\tau,1}$; and
- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]$ in $\mathsf{Hyb}_{\tau,2}$.

Without loss of generality, we assume that algorithm $\mathcal{A}$ always outputs a tuple $(C, P, (x_1, \ldots, x_K))$ where $P(\vec{\beta}) = 0$ and $\vec{\beta} = (\beta_1, \ldots, \beta_K)$ is as defined in Eq. (5.1). Otherwise, the output in both experiments is 0. Now, take any input $(i, c, \mathsf{Step}, x, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{step}}, \pi_{\mathsf{inst}}, w, \sigma)$ to these two programs. First, note that $i \in [K]$ which means
$$(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i-1, c) \neq (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}}).$$

Thus, by punctured correctness of Π_{PPRF}, Step 4 of $\mathsf{MapProg}_1$ behaves identically in the two cases. Thus, the two programs can only differ on Step 6. We consider the following two cases.

- Suppose $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i) \neq (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})$. Then by punctured correctness of Π_{PPRF}, the two programs behave identically.
- Suppose $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, i, c_i) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})$. In this case, $\tau_i = \tau_K = 1$. If $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) \neq 1$, then both programs output $\perp$. We focus on the case where $\mathsf{Reachable}_{\vec{\beta}, i-1}(c) = 1$. Next, recall that $\mathsf{hk}_{\mathsf{step}}$ and $\mathsf{hk}_{\mathsf{inst}}$ are somewhere statistically binding on index K. In addition, in both experiments, the challenger computes
$$(\mathsf{h}^*_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}^*_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K))$$

  Thus, with overwhelming probability over the choice of $\mathsf{hk}_{\mathsf{step}}$ and $\mathsf{hk}_{\mathsf{inst}}$:

  - If $\mathsf{Step} \neq \mathsf{Step}_K$, then $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{step}}, \mathsf{h}_{\mathsf{step}}, K, \mathsf{Step}, \pi_{\mathsf{step}}) = 0$.
  - If $x \neq x_K$, then $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, K, x, \pi_{\mathsf{inst}}) = 0$.

  Thus, if $\mathsf{Step} \neq \mathsf{Step}_K$, then both programs output $\perp$ with overwhelming probability (over the choice of $\mathsf{hk}_{\mathsf{step}}$). It suffices to consider the case where $\mathsf{Step} = \mathsf{Step}_K$. We consider two possibilities:

  - Suppose $C(x, w) = 1$ and $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) = 1$. As argued above, with overwhelming probability over the choice of $\mathsf{hk}_{\mathsf{inst}}$, this case occurs only if $x = x_K$. This means $C(x_K, w) = C(x, w) = 1$, and in particular, that $\beta_K = 1$. By construction of $\mathsf{MapProg}_1$, we have $c_i = \mathsf{Step}(c, 1) = \mathsf{Step}_K(c, 1)$. Since $\mathsf{Reachable}_{\vec{\beta}, K-1}(c) = 1$ and $\beta_K = 1$, by Definition 5.5 this means that
$$1 = \mathsf{Reachable}_{\vec{\beta}, K}(\mathsf{Step}(c, 1)) = \mathsf{Reachable}_{\vec{\beta}, K}(c_i) = \mathsf{Reachable}_{\vec{\beta}, K}(c_{\mathsf{acc}}),$$
    which contradicts the premise that $P(\vec{\beta}) = 0$.

– Suppose that either $C(x,w)=0$ or $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{inst}}, \mathsf{h}_{\mathsf{inst}}, i, x, \pi_{\mathsf{inst}}) \neq 1$. In this case, $c_{\mathsf{acc}} = c_i = \mathsf{Step}(c, 0)$. Again, since $\mathsf{Reachable}_{\vec{\beta},K-1}(c) = 1$, we can appeal to Definition 5.5 to conclude that

$$1 = \mathsf{Reachable}_{\vec{\beta},K}(\mathsf{Step}(c,0)) = \mathsf{Reachable}_{\vec{\beta},K}(c_i) = \mathsf{Reachable}_{\vec{\beta},K}(c_{\mathsf{acc}}),$$

which again contradicts the premise that $P(\vec{\beta}) = 0$.

We conclude that with overwhelming probability over the choice of $\mathsf{hk}_{\mathsf{inst}}$ and $\mathsf{hk}_{\mathsf{step}}$, this case does not happen unless the adversary $\mathcal{A}$ outputs $(C, P, \vec{x})$ where $P(\vec{\beta}) = 1$ (in which case, the advantage of $\mathcal{A}$ is 0).

We conclude that both pairs of programs compute identical functionality. The claim now follows by $i\mathcal{O}$ security and a standard hybrid argument. □

Claim 5.30. *Suppose Π_{PPRF} satisfies punctured pseudorandomness and $\lambda_{\mathsf{PRF}} \geq \lambda$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_{\tau,2}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\tau,3}(\mathcal{A})]| = \mathsf{negl}(\lambda)$.*

Proof. Follows by a similar argument as the proof of Claim 5.21. □

Claim 5.31. *Suppose Π_{PPRF} is correct, $i\mathcal{O}$ is secure, $\lambda_{\mathsf{obf}} \geq \lambda$, and $m \geq \lambda + \ell$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_{\tau,3}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{end}}(\mathcal{A})]| = \mathsf{negl}(\lambda)$.*

Proof. By the same argument as in the proof of Claim 5.29, the following two programs compute identical functionality:

- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]$ in $\mathsf{Hyb}_{\tau,3}$; and
- $\mathsf{MapProg}_1[C, K, \mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}, c_{\mathsf{init}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, \vec{\beta}, \tau]$ in $\mathsf{Hyb}_{\mathsf{end}}$.

We now show that the following two programs also compute identical functionality:

- $\mathsf{VerProg}_2[K, c_{\mathsf{acc}}, k^{(\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, K, c_{\mathsf{acc}})}, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}, z^*]$ in $\mathsf{Hyb}_{\tau,3}$; and
- $\mathsf{VerProg}_1[K, c_{\mathsf{acc}}, k, \mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}}]$ in $\mathsf{Hyb}_{\mathsf{end}}$.

Take any input $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$ to these two programs.

- Suppose $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) = (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$. By definition, $\mathsf{VerProg}_1$ always outputs $\perp$ in this case. Consider the behavior of $\mathsf{VerProg}_2$. In $\mathsf{Hyb}_{\tau,3}$, the challenger samples $z^* \xleftarrow{\mathrm{R}} \{0,1\}^m$. Since $m \geq \lambda + \ell$, with overwhelming probability over the choice of z, there does not exist a string $y \in \{0,1\}^\ell$ such that $\mathsf{G}(y) = z^*$. Thus, with overwhelming probability over the choice of z, $\mathsf{VerProg}_2$ will always output 0 in this case.
- Suppose $(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}) \neq (\mathsf{h}^*_{\mathsf{step}}, \mathsf{h}^*_{\mathsf{inst}})$. Then, by punctured correctness, both programs output 1 if and only if
$$\mathsf{G}(\sigma) = \mathsf{G}(\mathsf{F}(k, (\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, K, c_{\mathsf{acc}}))).$$

We conclude that with overwhelming probability over the choice of $z \xleftarrow{\mathrm{R}} \{0,1\}^m$, both programs compute identical functionality. The claim now follows by $i\mathcal{O}$ security and a standard hybrid argument. □

Lemma 5.27 now follows by combining Claims 5.28 to 5.31. □

Lemma 5.32 (Property 5). *For all $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_{\mathsf{end}}(\mathcal{A}) = 1] = 0$.*

Proof. The output in $\mathsf{Hyb}_{\mathsf{end}}$ is computed as $\mathsf{ObfVer}(\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma)$ where the challenger computes

$$(\mathsf{h}_{\mathsf{step}}, \pi_{\mathsf{step},1}, \ldots, \pi_{\mathsf{step},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{step}}, (\mathsf{Step}_1, \ldots, \mathsf{Step}_K))$$
$$(\mathsf{h}_{\mathsf{inst}}, \pi_{\mathsf{inst},1}, \ldots, \pi_{\mathsf{inst},K}) = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{inst}}, (x_1, \ldots, x_K)).$$

By construction,
$$\mathsf{VerProg}_1[K, c_{\mathsf{acc}}, k, \mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}](\mathsf{h}_{\mathsf{step}}, \mathsf{h}_{\mathsf{inst}}, \sigma) = 0$$
for all σ. As such, the output in this experiment is always 0, as required. □

Theorem 5.13 now follows from Lemma 5.12 together with Lemmas 5.14 to 5.16, 5.27 and 5.32. ◻

Theorem 5.33 (Succinctness)**.** *Suppose Π_{SSB} is succinct. Let D be a bound on the size of the circuit that computes the admissible set of reachable states associated with policies in $\mathcal{P}$ (see Definition 5.5),*[7] *Then, there exist universal polynomials* poly_1 *and* poly_2 *such that the size of the* crs *output by* $\mathsf{Setup}(1^\lambda, C, K)$ *and the size of the canonical witness σ output by* $\mathsf{Map}(\mathsf{crs}, \vec{x}, \vec{w})$ *are bounded by*

$$|\mathsf{crs}| \le \mathsf{poly}_1(\lambda, |C|, S, D, \log K)$$
$$|\sigma| \le \mathsf{poly}_2(\lambda).$$

Proof. For all $\lambda \in \mathbb{N}$, all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all Boolean policies $P \in \mathcal{P}$ where $P \colon \{0,1\}^K \to \{0,1\}$, and all $x_1, \ldots, x_K \in \{0,1\}^n$, $w_1, \ldots, w_K \in \{0,1\}^h$ such that

$$P(C(x_1, w_1), \ldots, C(x_K, w_K)) = 1,$$

the common reference string crs output by $\mathsf{Setup}(1^\lambda, C, K)$ consists of the following components:

- hash keys $\mathsf{hk}_{\mathsf{step}}, \mathsf{hk}_{\mathsf{inst}}$ for the somewhere-statistically-binding hash function;
- an obfuscated program ObfMap for generating proofs; and
- an obfuscated program ObfVer for verifying proofs.

Succinctness of Π_{SSB} ensures that $|\mathsf{hk}_{\mathsf{step}}| \le \mathsf{poly}(\lambda, s, \log K)$ and $|\mathsf{hk}_{\mathsf{inst}}| \le \mathsf{poly}(\lambda, n, \log K)$, where $s = \mathsf{poly}(\lambda, S)$ is a bound on the size of the Boolean circuit that computes a step function Step_i associated with a policy in $\mathcal{P}$. Next, we bound the sizes $\mathsf{size}_{\mathsf{MapProg}}$ and $\mathsf{size}_{\mathsf{VerProg}}$ of the obfuscated programs ObfMap and ObfVer (and their variants) that appear in the analysis of Theorem 5.13.

- First, the mapping programs MapProg appearing in the proof of Theorem 5.13 require hardwiring the $\mathsf{Reachable}_{\vec{\beta},i}$ circuit for all indices i where $\tau_i = 1$. From Lemma 5.12, for every $\tau \in \{0,1\}^K$ appearing in the analysis of Theorem 5.13, there are at most $1 + \log K$ indices $i \in [K]$ where $\tau_i = 1$. Thus, we can bound $\mathsf{size}_{\mathsf{MapProg}} = \mathsf{poly}(\lambda, |C|, S, D, \log K)$.
- Next, the sizes of verification programs VerProg appearing in the proof of Theorem 5.13 can be bounded by $\mathsf{size}_{\mathsf{VerProg}} = \mathsf{poly}(\lambda, n, S, \log K)$.

Taken together, we can bound the overall CRS size for Construction 5.8 by $\mathsf{poly}_1(\lambda, |C|, S, D, \log K)$. Next, the size of the canonical witness σ output by $\mathsf{Map}(\mathsf{crs}, (x_1, \ldots, x_K), (w_1, \ldots, w_K))$ is simply an ℓ-bit string, where ℓ is the seed length of a PRG. Here, we can take $\ell = \mathsf{poly}_2(\lambda)$, which completes the proof. ◻

Instantiation. Taken together, Construction 5.8 yields a succinct unique witness map for batch NP that supports any read-once bounded-space policy using $i\mathcal{O}$ together with a somewhere statistically binding hash function and an injective PRG. The somewhere statistically binding hash function and the injective PRG can be instantiated from standard number-theoretic assumptions [HW15, OPWW15, GKVW20]. We summarize our instantiation with the following corollary:

Corollary 5.34 (Succinct Unique Witness Map for Read-Once Bounded-Space Turing Machines)**.** *Let λ be a security parameter. Assuming the existence of indistinguishability obfuscation for Boolean circuits, a somewhere statistically binding hash function, and an injective PRG, there exists a succinct unique witness map for the set of monotone policies $\mathcal{P}$ that can be computed by a read-once bounded-space Turing machine. Specifically, if K is a bound on the number of instances, S is the space required by policies in P, and $D \le \mathsf{poly}(\lambda, 2^S)$ is the bound on the size of the circuit that computes the admissible set of reachable states associated with policies in $\mathcal{P}$, then the size of the CRS for the succinct unique witness map for a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and policy family $\mathcal{P}$ is* $\mathsf{poly}(\lambda, |C|, S, D, \log K)$ *and the proof size is* $\mathsf{poly}(\lambda)$.

[7]We can trivially bound $D = O(2^S)$ by enumerating the set of reachable states for the space-S Turing machine.

6 Applications

In this section, we highlight two immediate applications of succinct witness encryption for batch languages. The first application is to succinct computational secret sharing [ABI+23] and the second is to distributed monotone-policy encryption, a generalization of notions like distributed broadcast encryption [WQZD10, BZ14] and threshold encryption with silent setup [GKPW24, ADM+24].

6.1 Succinct Computational Secret Sharing

In this section, we show that succinct witness encryption for batch languages immediately implies a succinct computational secret sharing scheme. We start by recalling the notion of succinct computational secret sharing from [ABI+23].

Definition 6.1 (Succinct Computational Secret Sharing [ABI+23, adapted])**.** Let λ be a security parameter, $\mathcal{P}$ be a family of policies, and $\mathcal{M}$ be a message space. We model each policy $P \in \mathcal{P}$ as a monotone Boolean function. A succinct computational secret sharing scheme with policy space $\mathcal{P}$ is a pair of efficient algorithms $\Pi_{\mathsf{SCSS}} = (\mathsf{Share}, \mathsf{Reconstruct})$ with the following syntax:

- $\mathsf{Share}(1^\lambda, P, \mu) \to (\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n)$: On input the security parameter $\lambda \in \mathbb{N}$, a policy $P \in \mathcal{P}$ (on n-bit inputs), and a message $\mu \in \mathcal{M}$, the share algorithm outputs a collection of $n+1$ shares $\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n$, where sh_i is the share of the ith party and sh_0 is the public information given to all parties. By default, the public information sh_0 is an empty string.
- $\mathsf{Reconstruct}(P, \beta, \mathsf{sh}_0, \{(i, \mathsf{sh}_i)\}_{i \in [n]:\beta_i = 1}) \to \mu$: On input a policy $P \in \mathcal{P}$ (on n-bit inputs), a string $\beta \in \{0,1\}^n$ (describing the reconstructing set), the public information sh_0, and shares sh_i for indices $i \in [n]$ where $\beta_i = 1$, the reconstruction algorithm outputs a message $\mu \in \mathcal{M}$.

We require that Π_{SCSS} satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all policies $P \in \mathcal{P}$ (on n-bit inputs), all $\beta_1, \ldots, \beta_n \in \{0,1\}$ where $P(\beta_1, \ldots, \beta_n) = 1$, all messages $\mu \in \mathcal{M}$, we have that
$$\Pr[\mathsf{Reconstruct}(P, \beta, \mathsf{sh}_0, \{(i, \mathsf{sh}_i)\}_{i \in [n]:\beta_i = 1}) = \mu : (\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, P, \mu)] = 1.$$
We say the scheme satisfies (statistical) correctness if there is a negligible function such that the above holds with probability $1 - \mathsf{negl}(\lambda)$.
- **Security:** For a security parameter $\lambda \in \mathbb{N}$, a bit $b \in \{0,1\}$, and an adversary $\mathcal{A}$, we define the security experiment as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ chooses a policy $P \in \mathcal{P}$ (on n-bit inputs), bits $\beta_1, \ldots, \beta_n \in \{0,1\}$ where $P(\beta_1, \ldots, \beta_n) = 0$, and two messages $\mu_0, \mu_1 \in \mathcal{M}$.
  - The challenger computes $(\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, P, \mu_b)$ and gives the shares sh_0 and sh_i where $\beta_i = 1$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the output of the experiment.

  The secret sharing scheme is secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the security game.
- **Succinctness:** There exists a polynomial poly such that for all $\lambda \in \mathbb{N}$, policies $P \in \mathcal{P}$, and messages $\mu \in \mathcal{M}$, the size of the public information sh_0 and shares sh_i output by $(\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, P, \mu)$ satisfy the following:
$$|\mathsf{sh}_0|, |\mathsf{sh}_i| \le o(|P|) \cdot \mathsf{poly}(\lambda, \log n).$$

Construction 6.2 (Succinct Computational Secret Sharing from Succinct Witness Encryption)**.** Let λ be a security parameter, $\mathcal{P}$ be a family of monotone access policies, and $\mathcal{M}$ be a message space. Our construction of succinct computational secret sharing relies on the following primitives:

- Let $\Pi_{\mathsf{PKE}} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ be a public-key encryption scheme with message space $\{0,1\}$. Let $\rho = \rho(\lambda)$ be the number of bits of randomness the $\mathsf{PKE.Encrypt}$ algorithm takes.
- Let $\Pi_{\mathsf{WE}} = (\mathsf{WE.Encrypt}, \mathsf{WE.Preprocess}, \mathsf{WE.DecryptLocal})$ be a succinct witness encryption scheme for batch languages that supports local decryption (Definition 3.2) with message space $\mathcal{M}$ and policy family $\mathcal{P}$.

We construct a succinct computational secret sharing scheme with message space $\mathcal{M}$ and policy space $\mathcal{P}$ as follows:

- $\mathsf{Share}(1^\lambda, P, \mu)$: On input the security parameter $\lambda \in \mathbb{N}$, the policy $P \in \mathcal{P}$ (on n-bit inputs), and a message $\mu \in \mathcal{M}$, the share algorithm proceeds as follows:
  - Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. Then, for each $i \in [n]$, samples $r_i \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and let $\mathsf{ct}_i = \mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r_i)$.
  - Compute $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n), \mu)$, where $C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}, r)$ outputs 1 if $\mathsf{ct} = \mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r)$ and 0 otherwise.
  - Compute $(\mathsf{ht}_1, \ldots, \mathsf{ht}_n) \leftarrow \mathsf{WE.Preprocess}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n))$.
  - Output the shares $\mathsf{sh}_0 = (\mathsf{pk}, \mathsf{ct}_{\mathsf{WE}})$ and for each $i \in [n]$, $\mathsf{sh}_i = (\mathsf{ct}_i, \mathsf{ht}_i, r_i)$.
- $\mathsf{Reconstruct}(P, \beta, \mathsf{sh}_0, \{(i, \mathsf{sh}_i)\}_{i \in [n] : \beta_i = 1})$: On input the policy P (on n-bit inputs), a string $\beta \in \{0,1\}^n$, a public share $\mathsf{sh}_0 = (\mathsf{pk}, \mathsf{ct}_{\mathsf{WE}})$, and shares $\mathsf{sh}_i = (\mathsf{ct}_i, \mathsf{ht}_i, r_i)$ for each i where $\beta_i = 1$, the decryption algorithm outputs
$$\mu = \mathsf{WE.DecryptLocal}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pk}], \{(i, \mathsf{ht}_i, r_i)\}_{i \in [n] : \beta_i = 1}).$$

Theorem 6.3 (Correctness)**.** *If Π_{WE} is correct, then Construction 6.2 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, policy $P \in \mathcal{P}$ (on n-bit inputs), inputs $\beta_1, \ldots, \beta_n \in \{0,1\}$ where $P(\beta_1, \ldots, \beta_n) = 1$, and any message $\mu \in \{0,1\}$. Suppose we compute $(\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, P, \mu)$. Consider the value of $\mathsf{Reconstruct}(P, \beta, \mathsf{sh}_0, \{(i, \mathsf{sh}_i)\}_{i \in [n] : \beta_i = 1})$:

- By construction, $\mathsf{sh}_0 = (\mathsf{pk}, \mathsf{ct}_{\mathsf{WE}})$ where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and
$$\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n), \mu).$$
  Here, $\mathsf{ct}_i \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r_i)$. This means $C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}_i, r_i) = 1$ for all $i \in [n]$.
- Since $P(\beta_1, \ldots, \beta_n) = 1$, correctness of local decryption (Definition 3.2) implies that
$$\mathsf{WE.DecryptLocal}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pk}], \{(i, \mathsf{ht}_i, r_i)\}_{i \in [n] : \beta_i = 1}) = \mu,$$
  where $(\mathsf{ht}_1, \ldots, \mathsf{ht}_n) = \mathsf{Preprocess}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n))$. Correctness follows. $\square$

Theorem 6.4 (Security)**.** *If Π_{PKE} satisfies perfect correctness and CPA-security and Π_{WE} is secure, then Construction 6.2 is secure.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the security game. We begin by defining a sequence of hybrid experiments, each parameterized by a bit $b \in \{0,1\}$:

- $\mathsf{Hyb}_0^{(b)}$: This is the security experiment with the bit $b \in \{0,1\}$.
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by choosing a policy $P \in \mathcal{P}$ (on n-bit inputs), inputs $\beta_1, \ldots, \beta_n \in \{0,1\}$ where $P(\beta_1, \ldots, \beta_n) = 0$, and two messages $\mu_0, \mu_1 \in \mathcal{M}$.
  - The challenger starts by sampling $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. For each $i \in [n]$, the challenger samples $\mathsf{ct}_i \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, 1)$.
  - The challenger then computes $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n), \mu_b)$.

  – Next, the challenger computes $(\mathsf{ht}_1, \ldots, \mathsf{ht}_n) \leftarrow \mathsf{WE.Preprocess}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n))$. It then gives the shares $\mathsf{sh}_0 = (\mathsf{pk}, \mathsf{ct}_{\mathsf{WE}})$ and $\mathsf{sh}_i = (\mathsf{ct}_i, \mathsf{ht}_i, r_i)$ for all $i \in [n]$ where $\beta_i = 1$ to $\mathcal{A}$.
  – Finally, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$, except the challenger samples $\mathsf{ct}_i \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 0)$ for all i where $\beta_i = 0$.

We write $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ to denote the output distribution of an execution of experiment $\mathsf{Hyb}_i^{(b)}$ with adversary $\mathcal{A}$. We now analyze the hybrid distributions.

Lemma 6.5. *If Π_{PKE} is CPA-secure, then for all $b \in \{0, 1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some $b \in \{0, 1\}$ and non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the CPA-security game.

- At the beginning of the game, algorithm $\mathcal{B}$ receives the security parameter 1^λ and a public key pk.
- Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a policy $P \in \mathcal{P}$ (on n-bit inputs), inputs $\beta_1, \ldots, \beta_n \in \{0, 1\}$ where $P(\beta_1, \ldots, \beta_n) = 0$, and two messages $\mu_0, \mu_1 \in \mathcal{M}$.
- For each $i \in [n]$ where $\beta_i = 0$, algorithm $\mathcal{B}$ makes an encryption query on the pair of messages $(1, 0)$ to obtain the ciphertext ct_i. For $i \in [n]$ where $\beta_i = 1$, algorithm $\mathcal{B}$ samples $r_i \xleftarrow{\mathrm{R}} \{0, 1\}^\rho$ and computes $\mathsf{ct}_i \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r_i)$.
- The challenger then computes $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n), \mu_b)$.
- Next, the challenger computes $(\mathsf{ht}_1, \ldots, \mathsf{ht}_n) \leftarrow \mathsf{WE.Preprocess}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n))$. It then gives the shares $\mathsf{sh}_0 = (\mathsf{pk}, \mathsf{ct}_{\mathsf{WE}})$ and $\mathsf{sh}_i = (\mathsf{ct}_i, \mathsf{ht}_i, r_i)$ for all $i \in [n]$ where $\beta_i = 1$ to $\mathcal{A}$.
- At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which $\mathcal{B}$ also outputs.

By definition, the challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, so the distribution of pk is perfectly simulated. In the reduction, if the challenger responds with encryptions of 1 (i.e., $\mathsf{ct}_i \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, 1)$), then algorithm $\mathcal{B}$ perfectly simulates the distribution of $\mathsf{Hyb}_0^{(b)}$. Alternatively, if the challenger responds with encryptions of 0 (i.e., $\mathsf{ct}_i \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, 1)$), then algorithm $\mathcal{B}$ perfectly simulates the distribution of $\mathsf{Hyb}_1^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks CPA-security of Π_{PKE} with the same advantage ε. □

Lemma 6.6. *If Π_{PKE} satisfies perfect correctness and Π_{WE} is secure, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the witness encryption security game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ outputs a policy $P \in \mathcal{P}$ (on n-bit inputs), inputs $\beta_1, \ldots, \beta_n \in \{0, 1\}$ where $P(\beta_1, \ldots, \beta_n) = 0$, and two messages $\mu_0, \mu_1 \in \mathcal{M}$.
2. Algorithm $\mathcal{B}$ samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. Then, for each $i \in [n]$ where $\beta_i = 0$, algorithm $\mathcal{B}$ computes $\mathsf{ct}_i \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, 0)$. If $\beta_i = 1$, algorithm $\mathcal{B}$ samples $r_i \xleftarrow{\mathrm{R}} \{0, 1\}^\rho$ and computes $\mathsf{ct}_i \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r_i)$.
3. Algorithm $\mathcal{B}$ outputs the circuit $C_{\mathsf{ValidShare}}[\mathsf{pk}]$, the policy P, the statements $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$, and the messages μ_0, μ_1. The challenger responds with a ciphertext $\mathsf{ct}_{\mathsf{WE}}$.
4. Next, the challenger computes $(\mathsf{ht}_1, \ldots, \mathsf{ht}_n) \leftarrow \mathsf{WE.Preprocess}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n))$. It then gives the shares $\mathsf{sh}_0 = (\mathsf{pk}, \mathsf{ct}_{\mathsf{WE}})$ and $\mathsf{sh}_i = (\mathsf{ct}_i, \mathsf{ht}_i, r_i)$ for all $i \in [n]$ where $\beta_i = 1$ to $\mathcal{A}$.

5. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

First, we argue that for all $r_1, \ldots, r_n \in \{0,1\}^\rho$, we have $P(C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}_1, r_1), \ldots, C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}_n, r_n)) = 0$:

- First, for all $i \in [n]$ where $\beta_i = 0$, there does not exist $r \in \{0,1\}^\rho$ where $C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}_i, r) = 1$. By definition, $C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}_i, r)$ outputs 1 if and only if $\mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r) = \mathsf{ct}_i$. However, in this experiment, whenever $\beta_i = 0$, the challenger constructs ct_i to be an encryption of 0 under pk. Since Π_{PKE} satisfies perfect correctness, there does not exist any $r \in \{0,1\}^\rho$ where $\mathsf{ct}_i = \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1; r)$.
- Take any candidate witness $(r_1, \ldots, r_n)$. Let $\beta'_i = C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}_i, r_i)$. By the previous property, we have that $\beta'_i = 0$ whenever $\beta_i = 0$. This means that for all $i \in [n]$, $\beta'_i \le \beta_i$. Since P is monotone, this means that $P(\beta'_1, \ldots, \beta'_n) \le P(\beta_1, \ldots, \beta_n) = 0$, as required.

We conclude that

$$\forall r_1, \ldots, r_n \in \{0,1\}^\lambda : P(C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}_1, r_1), \ldots, C_{\mathsf{ValidShare}}[\mathsf{pk}](\mathsf{ct}_n, r_n)) = 0.$$

In this case, the witness encryption challenger either encrypts the message μ_0 or the message μ_1. If the challenger computes $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_0)$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(0)}$. Alternatively, if the challenger computes $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_1)$, then it perfectly simulates an execution of $\mathsf{Hyb}_1^{(1)}$. Thus, algorithm $\mathcal{B}$ breaks security of witness encryption with the same advantage ε. □

Security now follows by combining Lemmas 6.5 and 6.6. □

Remark 6.7 (Using Witness Encryption for Trapdoor NP Relations)**.** When the underlying public-key encryption scheme Π_{PKE} in Construction 6.2 satisfies an "encryption with randomness recovery" property (i.e., where the decryption algorithm recovers both the message and the encryption randomness) [HKW20], then the NP relation $C_{\mathsf{ValidShare}}[\mathsf{pk}]$ in Construction 6.2 is a trapdoor NP relation. Specifically, the trapdoor relation is the circuit $C[\mathsf{sk}]$ with the secret key sk hard-wired inside it. On input a ciphertext ct, the $C[\mathsf{sk}]$ circuit computes $(\mu, r) \leftarrow \mathsf{PKE.Decrypt}(\mathsf{sk}, \mathsf{ct})$ and outputs 1 if $\mathsf{ct} = \mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r)$ and 0 otherwise. By perfect correctness of Π_{PKE}, whenever $\mathsf{ct} = \mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r)$, then decryption recovers $(1, r)$ and $C[\mathsf{sk}](\mathsf{ct}) = 1$. Alternatively, if $\mathsf{ct} \ne \mathsf{PKE.Encrypt}(\mathsf{pk}, 1; r)$ for some $r \in \{0,1\}^\rho$, then $C[\mathsf{sk}](\mathsf{ct}) = 0$. Thus, we can instantiate Construction 6.2 with any succinct witness encryption scheme for batch languages that supports trapdoor NP relations (as opposed to arbitrary NP relations). Moreover, public-key encryption schemes with randomness recovery can be constructed from any injective trapdoor function (see [Yao82, HKW20]).

Remark 6.8 (Using a PRG instead of Public-Key Encryption)**.** We can also replace the public-key encryption scheme in Construction 6.2 with a (sufficiently-expanding) pseudorandom generator. For instance, suppose $\mathsf{G} \colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ is a secure PRG. Then, instead of taking the shares to be public-key encryptions ct_i of 1 (with the randomness as the secret key), we define the shares to be $t_i = \mathsf{G}(s_i)$ where $s_i \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ is the associated secret key. The relation $C_{\mathsf{ValidShare}}$ then checks that $t_i = \mathsf{G}(s_i)$. In the security proof, we switch the shares t_i for the honest parties (i.e., indices i where $\beta_i = 0$) to uniform random strings $t_i \xleftarrow{\mathrm{R}} \{0,1\}^{2\lambda}$. With overwhelming probability over the choice of t_i, there no longer exists $s_i \in \{0,1\}^\lambda$ such that $\mathsf{G}(s_i) = t_i$, which is sufficient to invoke security of witness encryption. This approach has the advantage that we can replace the public-key encryption scheme with a PRG, but the resulting NP relation is no longer a trapdoor NP relation (see Remark 6.7 for more discussion).

Succinct computational secret sharing in the random oracle model. Construction 6.2 assumes the underlying succinct witness encryption scheme supports local decryption. While our construction for CNF formulas (Construction 4.1) and for read-once bounded-space Turing machines (Construction 5.8) support this property, our construction for DNF formulas (Construction 4.12) does not. In Appendix A, we show a variant of Construction 6.2 that works with *any* succinct witness encryption scheme (without local decryption), but relies on the random oracle heuristic. Namely, we show that using the notion of trapdoor proof systems from [FWW23], we can construct a succinct computational secret sharing scheme from a succinct witness encryption, where the instances for the witness encryption scheme consist of uniform random strings (and thus, can have a public, compact representation in the random oracle model). We refer to Appendix A for the construction details and analysis.

6.2 Distributed Monotone-Policy Encryption

In this section, we describe another application of succinct witness encryption for constructing distributed monotone-policy encryption. As mentioned in Section 1.1, distributed monotone-policy encryption generalizes notions like distributed broadcast encryption [WQZD10, BZ14] and threshold encryption with silent setup [GKPW24, ADM+24] to arbitrary monotone policies. We give the notion here, and then show a simple construction from a succinct witness encryption scheme for batch languages.

Definition 6.9 (Distributed Monotone-Policy Encryption)**.** Let λ be a security parameter, $\mathcal{P}$ be a family of monotone access policies, and $\mathcal{M}$ be a message space. We model each policy $P \in \mathcal{P}$ as a monotone Boolean function. A distributed monotone-policy encryption scheme with policy space $\mathcal{P}$ is a tuple of efficient algorithms $\Pi_{\mathsf{DMPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a set of public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: On input the public parameters pp, the key-generation algorithm outputs a public key pk_i and a secret key sk_i.
- $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu) \to \mathsf{ct}$: On input the public parameters pp, a policy $P \in \mathcal{P}$ (on n-bit inputs), a list of n public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and a message $\mu \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct. Note that the input length n is determined by P and is *not* fixed by the scheme.
- $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{sk}_i)\}_{i \in T}, \mathsf{ct}) \to \mu$: On input the public parameters pp, a policy $P \in \mathcal{P}$ (on n-bit inputs), a list of n public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, a list of decryption keys sk_i for $i \in T$ where $T \subseteq [n]$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M}$.

Moreover, Π_{DMPE} should satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all policies $P \in \mathcal{P}$ (on n-bit inputs), inputs $\beta_1, \ldots, \beta_n \in \{0,1\}$ where $P(\beta_1, \ldots, \beta_n) = 1$, all messages $\mu \in \mathcal{M}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, any set of public keys pk_i for $i \in [n] \setminus T$ where $T = \{i \in [n] : \beta_i = 1\}$, we have that

$$\Pr\left[\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{sk}_i)\}_{i \in T}, \mathsf{ct}) = \mu : \begin{array}{c} \forall i \in T : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu) \end{array}\right] = 1.$$

- **Security:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the security game as follows:
  - **Setup:** At the beginning of the game, the challenger samples the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and initializes a counter $\mathsf{ctr} = 0$ and an (empty) list $C = \varnothing$. The list C is used to keep track of corrupted keys. The challenger gives pp to $\mathcal{A}$.
  - **Pre-challenge query phase:** The adversary can now make the following queries:
    * **Key-generation query:** In a key-generation query, the challenger increments the counter $\mathsf{ctr} = \mathsf{ctr}+1$ and then samples samples $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and responds with $\mathsf{pk}_{\mathsf{ctr}}$.
    * **Corruption query:** In a corruption query, the adversary specifies a counter value $\mathsf{ctr}' \leq \mathsf{ctr}$, and the challenger replies with $\mathsf{sk}_{\mathsf{ctr}'}$. The adversary adds ctr' to C.
  - **Challenge phase:** In the challenge phase, the adversary specifies a policy $P \in \mathcal{P}$ and a pair of messages (μ_0, μ_1). In addition, for each $i \in [n]$, algorithm $\mathcal{A}$ specifies a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where algorithm $\mathcal{A}$ specifies a counter value $\mathsf{ctr}_i \leq \mathsf{ctr}$, the challenger sets $\mathsf{pk}_i = \mathsf{pk}_{\mathsf{ctr}}$. The challenger replies to $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_b)$.
  - **Post-challenge query phase:** The adversary can continue to make corruption queries in this phase. (Note that post-challenge key-generation queries are not useful).
  - **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

Let $\beta_i = 1$ if the adversary specified a public key pk_i or a corrupted counter value ctr_i where $\mathsf{ctr}_i \in C$ during the challenge phase. For indices $i \in [n]$ where $\mathcal{A}$ specified an uncorrupted counter value $\mathsf{ctr}_i \notin C$, let $\beta_i = 0$. We say that $\mathcal{A}$ is admissible if $P(\beta_1, \ldots, \beta_n) = 0$. We say that Π_{DMPE} is secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$$

in the above security game.

- **Succinctness:** There exists a polynomial poly such that for all $\lambda \in \mathbb{N}$, public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, policies $P \in \mathcal{P}$ (on n-bit inputs), public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and messages $\mu \in \mathcal{M}$, the size of the ciphertext ct output by $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$ is $|\mu| + o(|P|) \cdot \mathsf{poly}(\lambda, \log n)$.

Definition 6.10 (Static Security)**.** Let Π_{DMPE} be a distributed monotone-policy encryption scheme with policy space $\mathcal{P}$. We say that Π_{DMPE} satisfies static security if Π_{DMPE} is secure against adversaries that does not make any corruption queries during the query phase. Note that the adversary is still allowed to choose public keys itself (for any non-accepting subset of parties) during the challenge phase.

Constructing monotone-policy encryption from succinct witness encryption. We can leverage a similar strategy as used in our construction of computational secret sharing (Construction 6.2) to construct a monotone-policy encryption scheme from succinct witness encryption together with any public-key encryption scheme. Much like the case for computational secret sharing (Remark 6.8), we can also replace the public-key encryption scheme with a PRG instead, though this will require us to instantiate the witness encryption scheme with one that supports general NP relations as opposed to trapdoor NP relations (see Remark 6.16).

Construction 6.11 (Distributed Monotone-Policy Encryption)**.** Let λ be a security parameter, $\mathcal{M}$ be a message space, and $\mathcal{P}$ be a family of monotone policies. Our construction of monotone-policy encryption relies on the following primitives:

- Let $\Pi_{\mathsf{PKE}} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ be a public-key encryption scheme.
- Let $\Pi_{\mathsf{WE}} = (\mathsf{WE.Encrypt}, \mathsf{WE.Decrypt})$ be a succinct witness encryption scheme for batch languages with message space $\mathcal{M}$ and policy family $\mathcal{P}$.

We construct a distributed monotone-policy encryption scheme $\Pi_{\mathsf{DMPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with message space $\mathcal{M}$ and policy space $\mathcal{P}$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter $\lambda \in \mathbb{N}$, sample $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and output $\mathsf{pp} = (1^\lambda, \mathsf{pk}_{\mathsf{PKE}})$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (1^\lambda, \mathsf{pk}_{\mathsf{PKE}})$, the key-generation algorithm samples $r \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and outputs the public key $\mathsf{pk} = \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1; r)$ and the secret key $\mathsf{sk} = r$.
- $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$: On input the public parameters $\mathsf{pp} = (1^\lambda, \mathsf{pk}_{\mathsf{PKE}})$, the policy $P \in \mathcal{P}$ (on n-bit inputs), a tuple of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and a message $\mu \in \mathcal{M}$, the encryption algorithm defines the Boolean circuit $C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}]$ to be the circuit that takes as input a statement ct and a witness $r \in \{0,1\}^\rho$ and outputs 1 if and only if $\mathsf{ct} = \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1; r)$. The encryption algorithm outputs the ciphertext

$$\mathsf{ct} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu).$$

- $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{sk}_i)\}_{i \in T}, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = (1^\lambda, \mathsf{pk}_{\mathsf{PKE}})$, a policy $P \in \mathcal{P}$ (on n-bit inputs), the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, a collection of secret keys $\{(i, \mathsf{sk}_i\}_{i \in T}$, and the ciphertext ct, the decryption algorithm first defines $\mathsf{sk}_i = 0^\rho$ for all $i \in [n] \setminus T$. Then it outputs the message

$$\mu = \mathsf{WE.Decrypt}(\mathsf{ct}, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)).$$

Theorem 6.12 (Correctness)**.** *If Π_{WE} is correct, then Construction 6.11 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, a policy $P \in \mathcal{P}$, bits $\beta_1, \ldots, \beta_n \in \{0,1\}$ where $P(\beta_1, \ldots, \beta_n) = 1$, a message $\mu \in \mathcal{M}$, any $\mathsf{pp} = (1^\lambda, \mathsf{pk}_{\mathsf{PKE}})$ in the support $\mathsf{Setup}(1^\lambda)$, and any collection of public keys pk_i for $i \in [n] \setminus T$, where $T = \{i \in [n] : \beta_i = 1\}$. Suppose we sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for all $i \in T$. Then, $\mathsf{pk}_i = \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1; \mathsf{sk}_i)$. Take $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$. This means $\mathsf{ct} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$. Consider now $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{sk}_i)\}_{i \in T}, \mathsf{ct})$:

- By definition Decrypt sets $\mathsf{sk}_i = 0^\rho$ for all $i \in [n] \setminus T$.
- Since $\mathsf{pk}_i = \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1; \mathsf{sk}_i)$ for all $i \in T$, we have that $C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}](\mathsf{pk}_i, \mathsf{sk}_i) = 1$ for all $i \in T$. This means $\beta_i \le C_{\mathsf{ValidKey}}(\mathsf{pk}_i, \mathsf{sk}_i)$ for all $i \in [n]$.
- Since P is monotone, this means $1 = P(\beta_1, \ldots, \beta_n) \le P(C_{\mathsf{ValidKey}}(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, C_{\mathsf{ValidKey}}(\mathsf{pk}_n, \mathsf{sk}_n))$. Thus $P(C_{\mathsf{ValidKey}}(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, C_{\mathsf{ValidKey}}(\mathsf{pk}_n, \mathsf{sk}_n)) = 1$.
- By correctness of Π_{WE}, the decryption algorithm outputs μ. □

Theorem 6.13 (Static Security)**.** *If Π_{PKE} satisfies perfect correctness and CPA-security and Π_{WE} satisfies semantic security, then Construction 6.11 is statically secure.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the static security game. We begin by defining a sequence of hybrid experiments, each parameterized by a bit $b \in \{0,1\}$:

- $\mathsf{Hyb}_0^{(b)}$: This is the static security experiment with the bit $b \in \{0,1\}$.
  - At the beginning of the game, the challenger initializes a counter $\mathsf{ctr} = 0$ and samples $\mathsf{pk}_{\mathsf{PKE}} \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. The challenger gives $\mathsf{pp} = (1^\lambda, \mathsf{pk}_{\mathsf{PKE}})$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ can now make key-generation queries to the challenger. On each query, the challenger increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$. Then it samples $r_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and replies to $\mathcal{A}$ with $\mathsf{pk}^*_{\mathsf{ctr}} = \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1; r_{\mathsf{ctr}})$.
  - In the challenge phase, algorithm $\mathcal{A}$ specifies (a policy) P and a pair of messages μ_0, μ_1. For each $i \in [n]$, algorithm $\mathcal{A}$ specifies either a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where $\mathcal{A}$ specified a counter value ctr_i, the challenger sets $\mathsf{pk}_i = \mathsf{pk}^*_{\mathsf{ctr}_i}$. The challenger replies with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_b)$.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$, except when responding to key-generation queries, the challenger samples $\mathsf{pk}^*_{\mathsf{ctr}} = \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 0; r_{\mathsf{ctr}})$.

We write $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ to denote the output distribution of an execution of experiment $\mathsf{Hyb}_i^{(b)}$ with adversary $\mathcal{A}$. We now analyze the hybrid distributions.

Lemma 6.14. *If Π_{PKE} is CPA-secure, then for all $b \in \{0,1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| \ge \varepsilon$ for some $b \in \{0,1\}$ and non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the CPA-security game.

1. At the beginning of the game, algorithm $\mathcal{B}$ receives the security parameter 1^λ and a public key $\mathsf{pk}_{\mathsf{PKE}}$. Algorithm $\mathcal{B}$ initializes a counter $\mathsf{ctr} = 0$.
2. Whenever $\mathcal{A}$ makes a key-generation query, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and then makes an encryption query on the pair of messages $(1, 0)$. The challenger replies with the ciphertext $\mathsf{pk}^*_{\mathsf{ctr}}$, which $\mathcal{B}$ forwards to $\mathcal{A}$.

3. In the challenge phase, algorithm $\mathcal{A}$ specifies a policy P and a pair of messages μ_0, μ_1. For each $i \in [n]$, it also specifies a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where $\mathcal{A}$ specified a counter value ctr_i, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i = \mathsf{pk}^*_{\mathsf{ctr}_i}$. Algorithm $\mathcal{B}$ then replies with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_b)$.

4. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which $\mathcal{B}$ also outputs.

By definition, the challenger samples $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, so the distribution of $\mathsf{pk}_{\mathsf{PKE}}$ is perfectly simulated. In the reduction, if the challenger responds with encryptions of 1 (i.e., $\mathsf{pk}^*_{\mathsf{ctr}} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1)$), then algorithm $\mathcal{B}$ perfectly simulates the distribution of $\mathsf{Hyb}_0^{(b)}$. Alternatively, if the challenger responds with encryptions of 0 (i.e., $\mathsf{pk}^*_{\mathsf{ctr}} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 0)$), then algorithm $\mathcal{B}$ perfectly simulates the distribution of $\mathsf{Hyb}_1^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks CPA-security of Π_{PKE} with the same advantage ε. □

Lemma 6.15. *If $\mathcal{A}$ is admissible, Π_{PKE} satisfies perfect correctness, and Π_{WE} is secure, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the witness encryption security game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ initializes a counter $\mathsf{ctr} = 0$.

2. Whenever $\mathcal{A}$ makes a key-generation query, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and replies with $\mathsf{pk}^*_{\mathsf{ctr}} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 0)$.

3. In the challenge phase, algorithm $\mathcal{A}$ specifies a policy P and a pair of messages μ_0, μ_1. For each $i \in [n]$, it also specifies a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where $\mathcal{A}$ specified a counter value ctr_i, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i = \mathsf{pk}^*_{\mathsf{ctr}_i}$. Algorithm $\mathcal{B}$ gives the circuit C_{ValidKey}, the policy P, the instances $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and the pair of messages μ_0, μ_1 to the challenger. The challenger responds with a ciphertext ct^*.

4. Algorithm $\mathcal{B}$ gives ct^* to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

First, we argue that for all $r_1, \ldots, r_n \in \{0,1\}^\rho$, we have $P(C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}](\mathsf{pk}_1, r_1), \ldots, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}](\mathsf{pk}_n, r_n)) = 0$:

- First, for all $j \in [\mathsf{ctr}]$, there does not exist $r \in \{0,1\}^\rho$ such that $C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}](\mathsf{pk}^*_j, r) = 1$. By definition, $C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}](\mathsf{pk}^*_j, r)$ outputs 1 if and only if $\mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1; r) = \mathsf{pk}^*_j$. However, in this experiment, the challenger constructs pk^*_j to be an encryption of 0 under $\mathsf{pk}_{\mathsf{PKE}}$. Since Π_{PKE} satisfies perfect correctness, there does not exist any $r \in \{0,1\}^\rho$ where $\mathsf{pk}^*_j = \mathsf{PKE.Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, 1; r)$.

- For each $i \in [n]$, let $\beta^*_i = 1$ if algorithm $\mathcal{A}$ chose the public key pk_i. Let $\beta^*_i = 0$ if algorithm $\mathcal{A}$ specified a counter value ctr_i for the ith public key (i.e., $\mathsf{pk}_i = \mathsf{pk}^*_j$ for some $j \in [\mathsf{ctr}]$). Since $\mathcal{A}$ is admissible, we have that $P(\beta^*_1, \ldots, \beta^*_n) = 0$.

- Take any candidate witness $(r_1, \ldots, r_n)$. Let $\beta_i = C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}](\mathsf{pk}_i, r_i)$. By the first property, we have that $\beta_i = 0 = \beta^*_i$ for all indices $i \in [n]$ where algorithm $\mathcal{A}$ specified a counter value. Since P is monotone, this means that $P(\beta_1, \ldots, \beta_n) \leq P(\beta^*_1, \ldots, \beta^*_n) = 0$, as required.

We conclude that

$$\forall r_1, \ldots, r_n \in \{0,1\}^\lambda : P(C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}](\mathsf{pk}_1, r_1), \ldots, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}](\mathsf{pk}_n, r_n)) = 0.$$

In this case, the witness encryption challenger either encrypts the message μ_0 or the message μ_1. If the challenger computes $\mathsf{ct}^* \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_0)$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(0)}$. Alternatively, if the challenger computes $\mathsf{ct}^* \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_1)$, then it perfectly simulates an execution of $\mathsf{Hyb}_1^{(1)}$. Thus, algorithm $\mathcal{B}$ breaks security of witness encryption with the same advantage ε. □

Static security now follows by combining Lemmas 6.14 and 6.15. □

Remark 6.16 (Using Witness Encryption for Trapdoor NP Relations)**.** Similar to the case for Construction 6.2 (see Remark 6.7), the relation $C_{\mathsf{ValidKey}}[\mathsf{pk}_{\mathsf{PKE}}]$ in Construction 6.11 is a trapdoor NP relation when we instantiate the underlying public-key encryption scheme with a scheme that supports randomness recovery.

One-round distributed decryption. The decryption process in Construction 6.11 takes all of the secret keys for users in an authorized set as input. In applications involving mutually distrusting parties, we would want to support decryption without requiring each individual party to reveal their individual secret key to other parties. One way to implement this is by having the parties run a multiparty computation protocol to evaluate the decryption algorithm. The ideal scenario in this setting would be a one-round protocol where each party takes the ciphertext, independently generates a "partial decryption" share, and then publishes their share. Afterwards, there is a public decoding algorithm that takes the partial decryption shares from any authorized set of parties and recovers the message. This type of one-round decryption is a common requirement in multiparty notions such as multi-key homomorphic encryption [MW16] and threshold encryption with silent setup [GKPW24, ADM+24].

In Appendix B, we describe a simple modification of Construction 6.11 to support this type of one-round decryption process. Namely, using a similar paradigm as [GKPW24, ADM+24], we replace each user's public key with a verification key for a signature scheme. Each ciphertext in turn has a tag, and the decryption keys associated with a *specific* ciphertext consist of signatures on the tag. The same decryption key (i.e., the signing key) can be used to generate partial decryption shares for arbitrarily-many ciphertexts (and moreover, the partial decryption shares for one ciphertext cannot be used to compromise security of a different ciphertext).

Acknowledgments

We thank Yuval Ishai for extensive discussions in an early phase of this project. Brent Waters is supported by NSF CNS-2318701, and a Simons Investigator award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptol. ePrint Arch.*, 2013.

[ABI+23] Benny Applebaum, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tianren Liu, and Vinod Vaikuntanathan. Succinct computational secret sharing. In *STOC*, 2023.

[ADM+24] Gennaro Avitabile, Nico Döttling, Bernardo Magri, Christos Sakkas, and Stella Wohnig. Signature-based witness encryption with compact ciphertext. In *ASIACRYPT*, 2024.

[AKY24] Shweta Agrawal, Simran Kumari, and Shota Yamada. Pseudorandom multi-input functional encryption and applications. *IACR Cryptol. ePrint Arch.*, 2024.

[AMYY25] Shweta Agrawal, Anuja Modi, Anshu Yadav, and Shota Yamada. Evasive LWE: attacks, variants & obfustopia. *IACR Cryptol. ePrint Arch.*, page 375, 2025.

[ASY22] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In *EUROCRYPT*, 2022.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, 2023.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, 2019.

[BDJ$^+$25] Pedro Branco, Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Pseudorandom obfuscation and applications. In *CRYPTO*, 2025.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4), 1989.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.

[BGI$^+$17] Saikrishna Badrinarayanan, Sanjam Garg, Yuval Ishai, Amit Sahai, and Akshay Wadia. Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In *ASIACRYPT*, 2017.

[BGL$^+$15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *STOC*, 2017.

[BISW18] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal SNARGs via linear multi-prover interactive proofs. In *EUROCRYPT*, 2018.

[BÜW24] Chris Brzuska, Akin Ünal, and Ivy K. Y. Woo. Evasive LWE assumptions: Definitions, classes, and counterexamples. In *ASIACRYPT*, 2024.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, 2011.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CGJ$^+$23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In *CRYPTO*, 2023.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In *STOC*, 2015.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In *CRYPTO*, 2025.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *CRYPTO*, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *FOCS*, 2021.

[CPW20] Suvradip Chakraborty, Manoj Prabhakaran, and Daniel Wichs. Witness maps and applications. In *PKC*, 2020.

[CPW23] Suvradip Chakraborty, Manoj Prabhakaran, and Daniel Wichs. A map of witness maps: New definitions and connections. In *PKC*, 2023.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In *CRYPTO*, 2018.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *STOC*, 1991.

[DWW24] Lalita Devadas, Brent Waters, and David J. Wu. Batching adaptively-sound snargs for NP. In *TCC*, 2024.

[FNV17] Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Predictable arguments of knowledge. In *PKC*, 2017.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In *CRYPTO*, 2023.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO*, 2013.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO*, 2024.

[GKVW20] Rishab Goyal, Venkata Koppula, Satyanarayana Vusirikala, and Brent Waters. On perfect correctness in (lockable) obfuscation. In *TCC*, 2020.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *FOCS*, 2017.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *CRYPTO*, 2014.

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In *EUROCRYPT*, 2017.

[GS18] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for turing machines. In *TCC*, 2018.

[GSWW22] Rachit Garg, Kristin Sheridan, Brent Waters, and David J. Wu. Fully succinct batch arguments for NP from indistinguishability obfuscation. In *TCC*, 2022.

[GVW19] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. Collusion resistant broadcast and trace from positional witness encryption. In *PKC*, 2019.

[HHY25] Tzu-Hsiang Huang, Wei-Hsiang Hung, and Shota Yamada. A note on obfuscation-based attacks on private-coin evasive LWE. *IACR Cryptol. ePrint Arch.*, page 421, 2025.

[HIJ$^+$16] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In *ITCS*, 2016.

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin. Lattice-based post-quantum io from circular security with random opening assumption (part II: zeroizing attacks against private-coin evasive LWE assumptions). In *CRYPTO*, 2025.

[HKW20] Susan Hohenberger, Venkata Koppula, and Brent Waters. Chosen ciphertext security from injective trapdoor functions. In *CRYPTO*, 2020.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, 2015.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, 2015.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, 2013.

[LMP24] Yanyi Liu, Noam Mazor, and Rafael Pass. On witness encryption and laconic zero-knowledge arguments. *IACR Cryptol. ePrint Arch.*, 2024.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT*, 2016.

[NWW24] Shafik Nassar, Brent Waters, and David J. Wu. Monotone policy BARGs from BARGs and additively homomorphic encryption. In *TCC*, 2024.

[NWW25] Shafik Nassar, Brent Waters, and David J. Wu. Monotone-policy BARGs and more from BARGs and quadratic residuosity. In *PKC*, 2025.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *ASIACRYPT*, 2015.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, 1999.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In *CRYPTO*, 2022.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In *ASIACRYPT*, 2022.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS*, 2010.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, 2022.

[WW24] Brent Waters and Daniel Wichs. Adaptively secure attribute-based encryption from witness encryption. In *TCC*, 2024.

[WW25a] Brent Waters and David J. Wu. A pure indistinguishability obfuscation approach to adaptively-sound SNARGs for NP. In *CRYPTO*, 2025.

[WW25b] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In *CRYPTO*, 2025.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *FOCS*, 2017.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, 1982.

[Zha16] Mark Zhandry. How to avoid obfuscation using witness PRFs. In *TCC*, 2016.

A Succinct Computational Secret Sharing in the Random Oracle Model

Our succinct computational secret sharing scheme from Section 6.1 (Construction 6.2) relies on a succinct witness encryption scheme that supports local decryption. Recall that in Construction 6.2, the public information is a succinct witness encryption ciphertext encrypted with respect to the statement $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$, where ct_i is part of the share for user i. In particular, each user only know their individual ct_i and not ct_j for $j \neq i$. Thus, when a set of users $S \subseteq [n]$ come together to reconstruct the secret, they only have ct_i for $i \in S$, and not ct_j for $j \notin S$. As such, reconstruction (i.e., the ability to decrypt the witness encryption ciphertext) critically relies on the ability to locally decrypt a ciphertext given knowledge of only a subset of the statements.

An alternative approach in the random oracle model. An alternative approach to relying on local decryption is to simply include the statement $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ as part of the public information sh_0. Of course, the scheme is no longer succinct in this case. However, suppose that the underlying scheme had the property where each ct_i is a uniform random string. In this case, we can rely on the random oracle heuristic to "compress" the statement $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$. Namely, we simply define $\mathsf{ct}_i \coloneqq H(i)$ where H is modeled as a random oracle. This would yield a succinct computational secret sharing scheme in the random oracle model from any succinct witness encryption scheme (without local decryption). In the context of Construction 6.2, each individual instance ct_i is an encryption of 1 under a public-key encryption scheme and the associated witness is the randomness r_i associated with ct_i. The problem is that if we simply replace ct_i with a uniform random string, then ct_i need not be in the support of the underlying encryption algorithm. In this case, there may not exist any randomness r_i that explains ct_i as an encryption of 1. Thus, we need to replace the public-key encryption scheme with a stronger notion.

Trapdoor proof generator. The work of [FWW23] encountered a similar issue when constructing an optimal broadcast encryption scheme from witness encryption (in the random oracle model). To bridge the gap, they introduced the notion of a trapdoor proof generator. We recall the notion from [FWW23, §5.1] here. A trapdoor proof generator is defined over a family of sets $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$. Using a trapdoor, it is possible to generate "proofs" π_x for elements $x \in \mathcal{X}$, and moreover, there is a *public* verification algorithm that verifies the proofs. Moreover, the public parameters pp of the trapdoor proof generator can be sampled in one of two (computationally) indistinguishable modes:

- **Normal mode:** In the normal mode, the trapdoor associated with pp can be used to generate proofs π for all but a negligible fraction of elements $x \in \mathcal{X}$.
- **Alternative mode:** In the alternative mode, the parameters pp effectively partitions $\mathcal{X}$ into two disjoint sets: a dense set $\mathcal{X}_T \subset \mathcal{X}$ and a sparse pseudorandom set $\mathcal{X}_F \subset \mathcal{X}$. In addition, there are two sampling algorithms: (1) $\mathsf{SampleTrue}$ which jointly samples an element $x \in \mathcal{X}_T$ together with an accepting proof π_x for x; and (2) $\mathsf{SampleFalse}$ which samples an element $x \in \mathcal{X}_F$ for which there does not exist any accepting proof π_x. Finally, the trapdoor in the alternative mode can be used to decide membership in $\mathcal{X}_T$ and $\mathcal{X}_F$.

We give the formal definition below:

Definition A.1 (Trapdoor Proof Generator [FWW23, Definition 5.4])**.** Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ be a sequence of efficiently-sampleable sets. A trapdoor proof generator for $\mathcal{X}$ is a tuple of polynomial-time algorithms $\Pi_{\mathsf{TPG}} = (\mathsf{Setup}, \mathsf{CreateProof}, \mathsf{Verify}, \mathsf{SetupAlt}, \mathsf{SampleTrue}, \mathsf{SampleFalse}, \mathsf{TDDecide})$ with the following properties:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})$: On input the security parameter λ, the setup algorithm outputs a set of public parameters pp and a trapdoor td. We assume that pp and td implicitly contain a description of λ.
- $\mathsf{CreateProof}(\mathsf{td}, x) \to \pi$: On input the trapdoor td and an instance $x \in \mathcal{X}_\lambda$, the proof creation algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{pp}, x, \pi) \to b$: On input the public parameters pp, an instance $x \in \mathcal{X}_\lambda$, and a proof π, the verification algorithm outputs a bit $b \in \{0, 1\}$.
- $\mathsf{SetupAlt}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})$: On input the security parameter λ, the alternative setup algorithm outputs a set of public parameters pp and a trapdoor td. We assume that pp and td implicitly contain a description of λ.

- $\mathsf{SampleTrue}(\mathsf{pp}) \to (x, \pi)$: On input the public parameters pp, this sampling algorithm outputs an instance $x \in \mathcal{X}_\lambda$ together with a proof π.
- $\mathsf{SampleFalse}(\mathsf{pp}) \to x$: On input the public parameters pp, this sampling algorithm outputs an instance $x \in \mathcal{X}_\lambda$.
- $\mathsf{TDDecide}(\mathsf{td}, x) \to b$: On input the trapdoor td and an instance $x \in \mathcal{X}_\lambda$, the decider algorithm outputs a bit $b \in \{0, 1\}$.

We require Π_{TPG} satisfy the following properties:

- **Correctness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 : \begin{array}{c} (\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda), x \xleftarrow{\mathrm{R}} \mathcal{X}_\lambda \\ \pi \leftarrow \mathsf{CreateProof}(\mathsf{td}, x) \end{array}\right].$$

- **Mode indistinguishability:** For an adversary $\mathcal{A}$, a security parameter λ, and a bit $b \in \{0, 1\}$, we define the mode indistinguishability experiment as follows:
  - If $b = 0$, the challenger starts by sampling $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$. Otherwise, the challenger samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda)$. The challenger gives pp to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ can now (adaptively) make sampling queries to the challenger:
    * **Sample true instance:** If $\mathcal{A}$ requests a true instance, then if $b = 0$, the challenger samples $x \xleftarrow{\mathrm{R}} \mathcal{X}_\lambda$ and $\pi \leftarrow \mathsf{CreateProof}(\mathsf{td}, x)$. If $b = 1$, the challenger samples $(x, \pi) \leftarrow \mathsf{SampleTrue}(\mathsf{pp})$. The challenger replies to $\mathcal{A}$ with (x, π).
    * **Sample false instance:** If $\mathcal{A}$ requests a false instance, then if $b = 0$, the challenger responds with $x \xleftarrow{\mathrm{R}} \mathcal{X}_\lambda$. If $b = 1$, then the challenger responds with $x \leftarrow \mathsf{SampleFalse}(\mathsf{pp})$.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  We say Π_{TPG} satisfies mode indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$

  in the above security game.

- **Trapdoor decidability:** The following properties hold:
  - **Accepting true instances:** For all (possibly unbounded) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 \wedge \mathsf{TDDecide}(\mathsf{td}, x) \neq 1 : \begin{array}{c} (\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] = \mathsf{negl}(\lambda).$$

  - **Rejecting false instances:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\mathsf{TDDecide}(\mathsf{td}, x) \neq 0 : \begin{array}{c} (\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda) \\ x \leftarrow \mathsf{SampleFalse}(\mathsf{pp}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Fact A.2 (Trapdoor Proof Generator [FWW23, §5.3])**.** Suppose there exists a public-key encryption scheme with pseudorandom ciphertexts and a computational non-interactive zero-knowledge (NIZK) proof system for NP. Then, there exists a trapdoor proof generator. In particular, there exists a trapdoor proof generator assuming polynomial hardness of the plain LWE assumption.

Succinct computational secret sharing. By substituting the trapdoor proof generator for the public-key encryption scheme in Construction 6.2, we obtain a computational secret sharing scheme where the public information sh_0 is a long uniform random string. This in turn yields a succinct computational secret sharing scheme in the random oracle model (where we derive the long uniform random string in sh_0 from the random oracle). In the following description, we describe our construction with a long uniform random string:

Construction A.3 (Succinct Computational Secret Sharing in the Random Oracle Model)**.** Let λ be a security parameter, $\mathcal{P}$ be a family of monotone access policies, and $\mathcal{M}$ be a message space. Our construction of succinct computational secret sharing relies on the following primitives:

- First, let $\Pi_{\mathsf{TPG}} = (\mathsf{TPG.Setup}, \mathsf{TPG.CreateProof}, \mathsf{TPG.Verify}, \mathsf{TPG.SetupAlt}, \mathsf{TPG.SampleTrue}, \mathsf{TPG.SampleFalse}, \mathsf{TPG.TDDecide})$ be a trapdoor proof generator over a set system $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$.
- Next, let $\Pi_{\mathsf{WE}} = (\mathsf{WE.Encrypt}, \mathsf{WE.Decrypt})$ be a succinct witness encryption scheme for batch languages with message space $\mathcal{M}$ and policy family $\mathcal{P}$.

We construct a succinct computational secret sharing scheme with message space $\mathcal{M}$ and policy space $\mathcal{P}$ as follows:

- $\mathsf{Share}(1^\lambda, P, \mu)$: On input the security parameter $\lambda \in \mathbb{N}$, the policy $P \in \mathcal{P}$ (on n-bit inputs), and a message $\mu \in \mathcal{M}$, the share algorithm proceeds as follows:
  - Sample $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{TPG.Setup}(1^\lambda)$. For each $i \in [n]$, sample $x_i \xleftarrow{\text{R}} \mathcal{X}_\lambda$ and $\pi_i \leftarrow \mathsf{TPG.CreateProof}(\mathsf{td}, x_i)$.
  - Compute $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pp}], P, (x_1, \ldots, x_n), \mu)$, where $C_{\mathsf{ValidShare}}[\mathsf{pk}](x, \pi)$ outputs 1 if $\mathsf{TPG.Verify}(\mathsf{pp}, x, \pi) = 1$ and 0 otherwise.
  - Output the public share $\mathsf{sh}_0 = (\mathsf{pp}, \mathsf{ct}_{\mathsf{WE}}, x_1, \ldots, x_n)$ and the individual shares $\mathsf{sh}_i = \pi_i$ for each $i \in [n]$. Note that since $x_1, \ldots, x_n$ are uniform random, we can also sample them as $(x_1, \ldots, x_n) \leftarrow H(\sigma)$ where $\sigma \xleftarrow{\text{R}} \{0,1\}^\lambda$ is a random seed and H is modeled as a random oracle. This yields a construction with succinct shares in the random oracle model.
- $\mathsf{Reconstruct}(P, \beta, \mathsf{sh}_0, \{(i, \mathsf{sh}_i)\}_{i \in [n] : \beta_i = 1})$: On input the policy P (on n-bit inputs), a string $\beta \in \{0,1\}^n$, a public share $\mathsf{sh}_0 = (\mathsf{pp}, \mathsf{ct}_{\mathsf{WE}}, x_1, \ldots, x_n)$, and shares $\mathsf{sh}_i = \pi_i$ for each i where $\beta_i = 1$, the decryption algorithm sets $w_i = \perp$ for all $i \in [n]$ where $\beta_i = 0$. Then it outputs
$$\mu = \mathsf{WE.Decrypt}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pp}], P, (x_1, \ldots, x_n), (w_1, \ldots, w_n)).$$

Theorem A.4 (Correctness)**.** *If Π_{WE} and Π_{TPG} are correct, then Construction A.3 is (statistically) correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, policy $P \in \mathcal{P}$ (on n-bit inputs), any input $\beta \in \{0,1\}^n$ where $P(\beta) = 1$, and any message $\mu \in \{0,1\}$. Suppose we compute $(\mathsf{sh}_0, \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, P, \mu)$. Consider the value of $\mathsf{Reconstruct}(P, \beta, \mathsf{sh}_0, \{(i, \mathsf{sh}_i)\}_{i \in [n] : \beta_i = 1})$:

- By construction, $\mathsf{sh}_0 = (\mathsf{pp}, x_1, \ldots, x_n)$ where $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{TPG.Setup}(1^\lambda)$ and
$$\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pp}], P, (x_1, \ldots, x_n), \mu).$$
Here, $x_i \xleftarrow{\text{R}} \mathcal{X}_\lambda$ and $\pi_i \leftarrow \mathsf{TPG.CreateProof}(\mathsf{td}, x_i)$. By completeness of Π_{TPG}, with overwhelming probability, $\mathsf{TPG.Verify}(\mathsf{pp}, x, \pi) = 1$. This means $C_{\mathsf{ValidShare}}[\mathsf{pp}](x_i, \pi_i) = 1$ for all $i \in [n]$.
- Since $P(\beta) = 1$, correctness of witness encryption now ensures that
$$\mathsf{WE.Decrypt}(\mathsf{ct}_{\mathsf{WE}}, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (x_1, \ldots, x_n), (\pi_1, \ldots, \pi_n)) = \mu. \qquad \square$$

Theorem A.5. *If Π_{WE} is semantically secure and Π_{TPG} satisfies mode indistinguishability and trapdoor decidability, then Construction A.3 is secure.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the security game. We begin by defining a sequence of hybrid experiments, each parameterized by a bit $b \in \{0,1\}$:

- $\mathsf{Hyb}_0^{(b)}$: This is the security experiment with the bit $b \in \{0,1\}$.
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by choosing a policy $P \in \mathcal{P}$ (on n-bit inputs), an input $\beta \in \{0,1\}^n$ where $P(\beta) = 0$, and two messages $\mu_0, \mu_1 \in \mathcal{M}$.
  - The challenger samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{TPG.Setup}(1^\lambda)$. For each $i \in [n]$, the challenger samples $x_i \xleftarrow{\mathrm{R}} \mathcal{X}_\lambda$ and $\pi_i \leftarrow \mathsf{TPG.CreateProof}(\mathsf{td}, x_i)$. Finally, the challenger computes

$$\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (x_1, \ldots, x_n), \mu_b).$$

  - The challenger gives the public information $\mathsf{sh}_0 = (\mathsf{pp}, x_1, \ldots, x_n)$ and the shares $\mathsf{sh}_i = \pi_i$ for each $i \in [n]$ where $\beta_i = 1$ to $\mathcal{A}$.
  - Finally, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$, except the challenger samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda)$. Then, for each $i \in [n]$, the challenger samples $x_i \leftarrow \mathsf{SampleFalse}(\mathsf{pp})$ if $\beta_i = 0$ and $(x_i, \pi_i) \leftarrow \mathsf{SampleTrue}(\mathsf{pp})$ if $\beta_i = 1$.

We write $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ to denote the output distribution of an execution of experiment $\mathsf{Hyb}_i^{(b)}$ with adversary $\mathcal{A}$. We now analyze the hybrid distributions.

Lemma A.6. *If Π_{TPG} satisfies mode indistinguishability, then for all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some $b \in \{0,1\}$ and non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the mode indistinguishability game.

- At the beginning of the game, algorithm $\mathcal{B}$ receives the security parameter 1^λ and the public parameters pp.
- Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a policy $P \in \mathcal{P}$ (on n-bit inputs), an input $\beta \in \{0,1\}^n$ where $P(\beta) = 0$, and two messages $\mu_0, \mu_1 \in \mathcal{M}$.
- For each $i \in [n]$ where $\beta_i = 0$, algorithm $\mathcal{B}$ requests a false instance x_i from the challenger. For each $i \in [n]$ where $\beta_i = 1$, algorithm $\mathcal{B}$ requests a true instance (x_i, π_i) from the challenger.
- The challenger then computes $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pk}], P, (x_1, \ldots, x_n), \mu_b)$.
- Next, the challenger gives the public information $\mathsf{sh}_0 = (\mathsf{pp}, x_1, \ldots, x_n)$ and the individual shares $\mathsf{sh}_i = \pi_i$ for all $i \in [n]$ where $\beta_i = 1$ to $\mathcal{A}$.[8]
- At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

We consider the two possibilities:

- Suppose the challenger samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{TPG.Setup}(1^\lambda)$, the true instances as $x_i \xleftarrow{\mathrm{R}} \mathcal{X}_\lambda$ and $\pi_i \leftarrow \mathsf{TPG.CreateProof}(\mathsf{td}, x_i)$, and the false instances as $x_i \xleftarrow{\mathrm{R}} \mathcal{X}_\lambda$. Then, algorithm $\mathcal{B}$ perfectly simulates the distribution in $\mathsf{Hyb}_0^{(b)}$.
- Conversely, suppose the challenger samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{TPG.SetupAlt}(1^\lambda)$, the true instances as $(x_i, \pi_i) \leftarrow \mathsf{TPG.SampleTrue}(\mathsf{pp})$, and the false instances as $x_i \leftarrow \mathsf{TPG.SampleFalse}(\mathsf{pp})$. Then, algorithm $\mathcal{B}$ perfectly simulates the distribution in $\mathsf{Hyb}_1^{(b)}$.

We conclude that algorithm $\mathcal{B}$ breaks mode indistinguishability of Π_{TPG} with the same advantage ε. □

[8]In the case where the instances are derived from the random oracle as $(x_1, \ldots, x_n) = H(\sigma)$ where $\sigma \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, the reduction algorithm in this step would program $H(\sigma)$ to the value of $(x_1, \ldots, x_n)$ obtained from the challenger.

Lemma A.7. *If Π_{TPG} satisfies trapdoor indistinguishability and Π_{WE} is secure, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the witness encryption security game:

- On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ outputs a policy $P \in \mathcal{P}$ (on n-bit inputs), an input $\beta \in \{0,1\}^n$ where $P(\beta) = 0$, and two messages $\mu_0, \mu_1 \in \mathcal{M}$.
- Algorithm $\mathcal{B}$ samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda)$. Then, for each $i \in [n]$ where $\beta_i = 0$, algorithm $\mathcal{B}$ samples $x_i \leftarrow \mathsf{TPG.SampleFalse}(\mathsf{pp})$. If $\beta_i = 1$, algorithm $\mathcal{B}$ samples $(x_i, \pi_i) \leftarrow \mathsf{TPG.SampleTrue}(\mathsf{pp})$.
- Algorithm $\mathcal{B}$ outputs the circuit $C_{\mathsf{ValidShare}}[\mathsf{pp}]$, the policy P, the statements $(x_1, \ldots, x_n)$, and the messages μ_0, μ_1. The challenger responds with a ciphertext $\mathsf{ct}_{\mathsf{WE}}$.
- Next, the challenger gives the public information $\mathsf{sh}_0 = (\mathsf{pp}, x_1, \ldots, x_n)$ and the shares $\mathsf{sh}_i = \pi_i$ for all $i \in [n]$ where $\beta_i = 1$ to $\mathcal{A}$.
- At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

First, we argue that for all $\pi_1, \ldots, \pi_n \in \{0,1\}^*$, we have $P(C_{\mathsf{ValidShare}}[\mathsf{pp}](x_1, \pi_1), \ldots, C_{\mathsf{ValidShare}}[\mathsf{pp}](x_n, \pi_n)) = 0$:

- First, for all $i \in [n]$ where $\beta_i = 0$, we claim that with overwhelming probability over the choice of pp and x_i, there does not exist π_i where $C_{\mathsf{ValidShare}}[\mathsf{pp}](x_i, \pi_i) = 1$. First, in $\mathsf{Hyb}_1^{(0)}$ and $\mathsf{Hyb}_1^{(1)}$, the challenger samples $x_i \leftarrow \mathsf{TPG.SampleFalse}(\mathsf{pp})$. BY the trapdoor decidability property, with overwhelming probability (over the choice of pp and x_i), it will be the case that $\mathsf{TPG.TDDecide}(\mathsf{td}, x_i) \neq 0$. Suppose moreover that there exists π_i such that $\mathsf{TPG.Verify}(\mathsf{pp}, x_i, \pi_i) = 1$. Again by trapdoor decidability, this means that with overwhelming probability (over the choice of pp), $\mathsf{TPG.TDDecide}(\mathsf{td}, x_i) \neq 1$. Thus, with overwhelming probability, this means $\mathsf{TDDecide}(\mathsf{td}, x_i) \notin \{0,1\}$, which is a contradiction. Hence, we conclude that there does not exist π_i such that $\mathsf{TPG.Verify}(\mathsf{pp}, x_i, \pi_i) = 1$.
- Take any candidate witness $(\pi_1, \ldots, \pi_n)$. Let $\beta_i' = C_{\mathsf{ValidShare}}[\mathsf{pk}](x_i, \pi_i)$. By the previous property, we have that $\beta_i' = 0$ whenever $\beta_i = 0$. This means that for all $i \in [n]$, $\beta_i' \leq \beta_i$. Since P is monotone, this means that $P(\beta_1', \ldots, \beta_n') \leq P(\beta_1, \ldots, \beta_n) = 0$, as required.

We conclude that with overwhelming probability over the choice of pp and x_i (where $\beta_i = 0$),

$$\forall \pi_1, \ldots, \pi_n \in \{0,1\}^* : P(C_{\mathsf{ValidShare}}[\mathsf{pp}](\mathsf{ct}_1, \pi_1), \ldots, C_{\mathsf{ValidShare}}[\mathsf{pp}](\mathsf{ct}_n, \pi_n)) = 0.$$

In this case, the witness encryption challenger either encrypts the message μ_0 or the message μ_1. If the challenger computes $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pp}], P, (x_1, \ldots, x_n), \mu_0)$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(0)}$. Alternatively, if the challenger computes $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidShare}}[\mathsf{pp}], P, (x_1, \ldots, x_n), \mu_1)$, then it perfectly simulates an execution of $\mathsf{Hyb}_1^{(1)}$. Thus, algorithm $\mathcal{B}$ breaks security of witness encryption with the same advantage ε. □

Security now follows by combining Lemmas A.6 and A.7. □

Remark A.8 (Using Witness Encryption for Trapdoor NP Relations)**.** We note that the NP relation $C_{\mathsf{ValidShare}}$ from Construction A.3 is a trapdoor NP relation. Namely, the trapdoor relation is the circuit $C[\mathsf{td}]$ with the trapdoor hard-wired inside it. On input an input x, the circuit simply outputs $\mathsf{TPG.TDDecide}(\mathsf{td}, x)$. When the scheme parameters are sampled in alternative mode $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{TPG.SetupAlt}(1^\lambda)$ as in the proof of Lemma A.7, the trapdoor decidability properties on Π_{TPG} coincide with the requirements for a trapdoor NP relation. This allows us to combine Construction A.3 with our succinct witness encryption for DNFs (Construction 4.12) to obtain a computational secret sharing scheme for DNF policies in the random oracle model where the share size scales with size of a single min-term (and polylogarithmically with the number of min-terms).

B Monotone-Policy Encryption with One-Round Distributed Decryption

In this section, we show how to extend our monotone-policy encryption scheme to support a one-round distributed decryption process. As discussed in Section 6.2, our goal is a scheme where each party takes the ciphertext, independently generates a decryption share, and then publishes their share. Then there is is a decoding algorithm that takes the partial decryption shares from any authorized set of parties and recovers the message. The first security requirement is that any unauthorized set of decryption shares for a given ciphertext should not leak any information about the associated message. In addition, *any* collection decryption shares for a ciphertext ct should not leak any information about a different ciphertext ct′. We now give the formal definition:

Definition B.1 (Distributed Monotone-Policy Encryption with 1-Round Decryption)**.** Let λ be a security parameter, $\mathcal{P}$ be a family of monotone access policies, and $\mathcal{M}$ be a message space. We model each policy $P \in \mathcal{P}$ as a monotone Boolean function. A distributed monotone-policy encryption scheme that supports 1-round decryption with policy space $\mathcal{P}$ is a tuple of efficient algorithms $\Pi_{\mathsf{DMPE\text{-}1R}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a set of public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: On input the public parameters pp, the key-generation algorithm outputs a public key pk_i and a secret key sk_i.
- $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu) \to \mathsf{ct}$: On input the public parameters pp, a policy $P \in \mathcal{P}$ (on n-bit inputs), a list of n public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and a message $\mu \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct. Note that the input length n is determined by P and is *not* fixed by the scheme.
- $\mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}) \to \mathsf{ht}_i$: On input the public parameters pp, a decryption key sk, and a ciphertext ct, the hint-computation algorithm outputs a decryption hint ht_i.
- $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct}) \to \mu$: On input the public parameters pp, a policy $P \in \mathcal{P}$ (on n-bit inputs), a list of n public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, a list of decryption hints ht_i for $i \in T$ where $T \subseteq [n]$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M}$.

Moreover, $\Pi_{\mathsf{DMPE\text{-}1R}}$ should satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all policies $P \in \mathcal{P}$ (on n-bit inputs), all inputs $\beta \in \{0,1\}^n$ where $P(\beta) = 1$, all messages $\mu \in \mathcal{M}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, any set of public keys pk_i for $i \in [n] \setminus T$ where $T = \{i \in [n] : \beta_i = 1\}$, we have that

$$\Pr\left[\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct}) = \mu : \begin{array}{c} \forall i \in T : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu) \\ \forall i \in T : \mathsf{ht}_i \leftarrow \mathsf{GetHint}(\mathsf{pp}, (i, \mathsf{sk}_i), \mathsf{ct}) \end{array}\right] = 1.$$

- **Security:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the security game as follows:
  - **Setup:** At the beginning of the game, the challenger samples the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and initializes a counter $\mathsf{ctr} = 0$ and an (empty) list $C = \varnothing$. The list C is used to keep track of corrupted keys. The challenger gives pp to $\mathcal{A}$.
  - **Pre-challenge query phase:** The adversary can now make the following queries:
    - **Key-generation query:** In a key-generation query, the challenger increments the counter $\mathsf{ctr} = \mathsf{ctr}+1$ and then samples samples $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and responds with $\mathsf{pk}_{\mathsf{ctr}}$.
    - **Corruption query:** In a corruption query, the adversary specifies a counter value $\mathsf{ctr}' \leq \mathsf{ctr}$, and the challenger replies with $\mathsf{sk}_{\mathsf{ctr}'}$. The challenger also adds ctr' to C.
    - **Hint-computation query:** In a hint-computation query, the adversary specifies a ciphertext ct and a counter $\mathsf{ctr}' \leq \mathsf{ctr}$. The challenger replies with $\mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_{\mathsf{ctr}'}, \mathsf{ct})$.

- **Challenge phase:** In the challenge phase, the adversary specifies a policy $P \in \mathcal{P}$ and a pair of messages (μ_0, μ_1). In addition, for each $i \in [n]$, algorithm $\mathcal{A}$ specifies a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where algorithm $\mathcal{A}$ specifies a counter value $\mathsf{ctr}_i \le \mathsf{ctr}$, the challenger sets $\mathsf{pk}_i = \mathsf{pk}_{\mathsf{ctr}}$. The challenger replies to $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \dots, \mathsf{pk}_n), \mu_b)$.
- **Post-challenge query phase:** The adversary can continue to make corruption and hint-computation queries. If the adversary requests the decryption hint for a counter ctr' and the challenge ciphertext ct^*, the challenger also adds ctr' to C.
- **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

Let $\beta_i = 1$ if the adversary specified a public key pk_i or if it chose a counter value ctr_i where $\mathsf{ctr}_i \in C$ during the challenge phase (where C is the corrupted set at the very end of the security game). For indices $i \in [n]$ where $\mathcal{A}$ specified an uncorrupted counter value $\mathsf{ctr}_i \notin C$, let $\beta_i = 0$. We say that $\mathcal{A}$ is admissible if $P(\beta) = 0$. We say that $\Pi_{\mathsf{DMPE\text{-}1R}}$ is secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda).$$

- **Succinctness:** There exists a polynomial poly such that for all $\lambda \in \mathbb{N}$, public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, policies $P \in \mathcal{P}$ (on n-bit inputs), public keys $\mathsf{pk}_1, \dots, \mathsf{pk}_n$, and messages $\mu \in \mathcal{M}$, the size of the ciphertext ct output by $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \dots, \mathsf{pk}_n), \mu)$ is $|\mu| + o(|P|) \cdot \mathsf{poly}(\lambda, \log n)$.

Definition B.2 (Static Security)**.** Let $\Pi_{\mathsf{DMPE\text{-}1R}}$ be a distributed monotone-policy encryption with 1-round decryption and policy space $\mathcal{P}$. For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the static security game to be the standard security game from Definition B.1, except on each key-generation query, the adversary must pre-declare whether the particular key will be corrupted or not. Specifically, on each key-generation query, the adversary additionally specifies a bit $\gamma \in \{0,1\}$. Let $\gamma_1, \dots, \gamma_{\mathsf{ctr}} \in \{0,1\}$ be the bits associated with the key-generation queries the adversary makes. The adversary is admissible for the static security game if the following conditions hold:

- For all corrupted indices $\mathsf{ctr} \in C$, we have $\gamma_{\mathsf{ctr}} = 1$.
- Let $\beta_i = 1$ if the adversary specified a public key pk_i or if it chose a counter value ctr_i where $\gamma_{\mathsf{ctr}_i} = 1$ during the challenge phase. For indices $i \in [n]$ where $\mathcal{A}$ specified an (uncorrupted) counter value ctr_i where $\gamma_{\mathsf{ctr}_i} = 0$, let $\beta_i = 0$. We require that $P(\beta_1, \dots, \beta_n) = 0$.

We say that $\Pi_{\mathsf{DMPE\text{-}1R}}$ is statically secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda).$$

Note that in this game, the adversary is allowed to make pre-challenge hint-computation queries with respect to *any* public key $\mathsf{pk}_{\mathsf{ctr}}$ output by a key-generation query (irrespective of the value of the bit γ_{ctr}).

Puncturable signatures. Our construction relies on (strongly) puncturable signatures (also known as "all-but-one signatures") [GVW19, ADM+24]. We begin by recalling the definition:

Definition B.3 (Puncturable Signature [GVW19, ADM+24, adapted])**.** A strongly puncturable (or all-but-one) signature scheme with message space $\mathcal{M} = \{\{0,1\}^{\rho(\lambda)}\}_{\lambda \in \mathbb{N}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{SPS}} = (\mathsf{KeyGen}, \mathsf{KeyGenP}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm outputs a key pair $(\mathsf{vk}, \mathsf{sk})$.
- $\mathsf{KeyGenP}(1^\lambda, m^*) \to (\mathsf{vk}, \mathsf{sk})$: On input a security parameter λ and a message $m^* \in \{0,1\}^\rho$, the punctured-key-generation algorithm outputs a key pair $(\mathsf{vk}, \mathsf{sk})$.

- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: On input a signing key sk and a message $m \in \{0,1\}^\rho$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to b$: On input a verification key vk, a message $m \in \{0,1\}^\rho$, and a signature σ, the verification algorithm outputs a bit $b \in \{0,1\}$.

Moreover, the puncturable signature scheme should satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$ and all $m \in \{0,1\}^\rho$, it holds that
$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1 \;:\; \begin{array}{c} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{array} \right] = 1.$$
- **Punctured correctness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $m^* \in \{0,1\}^\rho$, it holds that
$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1 \text{ for some } \sigma^* \in \{0,1\}^* : (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGenP}(1^\lambda, m^*)\right] = \mathsf{negl}(\lambda).$$
- **Punctured key indistinguishability:** For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the punctured key indistinguishability experiment as follows:
  1. On input a security parameter λ, the adversary $\mathcal{A}$ outputs a message $m^* \in \{0,1\}^\rho$ and sends it to the challenger.
  2. If $b = 0$, the challenger samples $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. If $b = 1$, the challenger samples $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGenP}(1^\lambda, m^*)$. It gives vk to $\mathcal{A}$.
  3. The adversary $\mathcal{A}$ can now make signing queries on messages $m \in \{0,1\}^\rho \setminus \{m^*\}$. On each signing query, the challenger replies with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$.
  4. The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{SPS} satisfies punctured key indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that
$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$$
  in the punctured key indistinguishability experiment.

Distributed monotone-policy encryption construction. We now show how to construct a statically-secure distributed monotone-policy encryption scheme that supports 1-round decryption using a succinct witness encryption scheme for batch languages. Our construction is a variant of Construction 6.11 where we replace the public-key encryption scheme with a puncturable signature scheme. Our approach leverages a similar strategy as that used to construct the succinct signature-based witness encryption scheme from [ADM+24]. Namely, a user's public key pk consists of a signature verification keys and their secret key sk is the associated signing key. An encryption of a message m with respect to user public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ and decryption policy P consists of a (random) tag τ together with a witness encryption $\mathsf{ct}_{\mathsf{WE}}$ of the message with respect to $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and the policy P. Decryption is only possible if one holds signatures on the tag τ that verify with respect to an authorized subset of public keys (i.e., a set of public keys that satisfy the access policy P). In this case, the decryption hints for any user is the signature on the tag τ. In the following construction, we take the tag τ to be the verification key $\mathsf{vk}_{\mathsf{OTS}}$ for a (one-time) signature scheme and include a signature on $\mathsf{ct}_{\mathsf{WE}}$ that verifies with respect to $\mathsf{vk}_{\mathsf{OTS}}$. Using a one-time signature enforces a non-malleability property on the ciphertexts, and is useful for supporting post-challenge hint queries. This is a standard approach that is commonly used in the setting of building non-malleable and CCA-secure public-key encryption [DDN91, Sah99]. We give the formal construction below:

Construction B.4 (Distributed Monotone-Policy Encryption with 1-Round Decryption)**.** Let λ be a security parameter, $\mathcal{M}$ be a message space, and $\mathcal{P}$ be a family of monotone policies. Our construction of distributed monotone-policy encryption with 1-round decryption relies on the following properties:

- Let $\Pi_{\mathsf{OTS}} = (\mathsf{OTS.KeyGen}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$ be a one-time signature scheme. Let $\rho = \rho(\lambda)$ be the length of the verification key output by Π_{OTS}.
- Let $\Pi_{\mathsf{SPS}} = (\mathsf{SPS.KeyGen}, \mathsf{SPS.KeyGenP}, \mathsf{SPS.Sign}, \mathsf{SPS.Verify})$ be a strongly puncturable signature scheme with message space $\{0,1\}^\rho$. Let $\kappa = \kappa(\lambda)$ be the signature size.
- Let $\Pi_{\mathsf{WE}} = (\mathsf{WE.Encrypt}, \mathsf{WE.Decrypt})$ be a succinct witness encryption scheme for batch languages with message space $\mathcal{M}$ and policy family $\mathcal{P}$.

We construct a distributed monotone-policy encryption scheme with 1-round decryption $\Pi_{\mathsf{DMPE\text{-}1R}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm outputs $\mathsf{pp} = 1^\lambda$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = 1^\lambda$, the key-generation algorithm samples $(\mathsf{vk}_{\mathsf{SPS}}, \mathsf{sk}_{\mathsf{SPS}}) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda)$. It outputs the public key $\mathsf{pk} = \mathsf{vk}_{\mathsf{SPS}}$ and the secret key $\mathsf{sk}_{\mathsf{SPS}}$.
- $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$: On input the public parameters $\mathsf{pp} = 1^\lambda$, the policy $P \in \mathcal{P}$ (on n-bit inputs), a tuple of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and a message $\mu \in \mathcal{M}$, the encryption algorithm proceeds as follows:
  - Sample a key-pair $(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{sk}_{\mathsf{OTS}}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$ for a one-time signature scheme
  - Define the Boolean circuit $C_{\mathsf{ValidSig}}[\mathsf{vk}_{\mathsf{OTS}}]$ to be the circuit that takes as input a statement $\mathsf{vk}_{\mathsf{SPS}}$ and a witness $\sigma_{\mathsf{SPS}} \in \{0,1\}^\kappa$ and outputs 1 if and only if $\mathsf{SPS.Verify}(\mathsf{vk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{OTS}}, \sigma_{\mathsf{SPS}}) = 1$.
  - Compute the ciphertext $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidSig}}[\mathsf{vk}_{\mathsf{OTS}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$ and the signature $\sigma_{\mathsf{OTS}} \leftarrow \mathsf{OTS.Sign}(\mathsf{sk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}})$

  Finally, it outputs the ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$.
- $\mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = 1^\lambda$, the secret key $\mathsf{sk} = \mathsf{sk}_{\mathsf{SPS}}$, and a ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, the hint-computation algorithm first checks that $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$ and outputs $\perp$ if not. Otherwise, it outputs $\mathsf{ht} = \sigma_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{SPS}}, \mathsf{vk}_{\mathsf{OTS}})$.
- $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = 1^\lambda$, a policy $P \in \mathcal{P}$ (on n-bit inputs), the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, a collection of hints $\{(i, \mathsf{ht}_i)\}_{i \in T}$, and the ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, the decryption algorithm first defines $\mathsf{ht}_i = 0^\kappa$ for all $i \in [n] \setminus T$. Then it outputs the message

$$\mu = \mathsf{WE.Decrypt}(\mathsf{ct}, C_{\mathsf{ValidSig}}[\mathsf{vk}_{\mathsf{OTS}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), (\mathsf{ht}_1, \ldots, \mathsf{ht}_n)).$$

Theorem B.5 (Correctness)**.** *If Π_{OTS}, Π_{SPS}, and Π_{WE} are correct, then Construction B.4 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, a policy $P \in \mathcal{P}$, input bits $\beta_1, \ldots, \beta_n \in \{0,1\}^n$ where $P(\beta_1, \ldots, \beta_n) = 1$, a message $\mu \in \mathcal{M}$, and any collection of public keys pk_i for $i \in [n] \setminus T$, where $T = \{i \in [n] : \beta_i = 1\}$. Let $\mathsf{pp} = 1^\lambda$. Suppose we sample the following quantities:

- Sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for all $i \in T$. By definition, this means $\mathsf{pk}_i = \mathsf{pk}_{\mathsf{SPS},i}$ and $\mathsf{sk}_i = \mathsf{sk}_{\mathsf{SPS},i}$ where $(\mathsf{vk}_{\mathsf{SPS},i}, \mathsf{sk}_{\mathsf{SPS},i}) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda)$.
- Take $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$. This means $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$ where

$$\begin{aligned} (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{sk}_{\mathsf{OTS}}) &\leftarrow \mathsf{OTS.KeyGen}(1^\lambda) \\ \mathsf{ct}_{\mathsf{WE}} &\leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidSig}}[\mathsf{vk}_{\mathsf{OTS}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu) \\ \sigma_{\mathsf{OTS}} &\leftarrow \mathsf{OTS.Sign}(\mathsf{sk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}). \end{aligned}$$

- Let $\mathsf{ht}_i \leftarrow \mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct})$ for each $i \in T$. By correctness of Π_{OTS}, $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$. Thus, for all $i \in [T]$, $\mathsf{ht}_i = \sigma_{\mathsf{SPS},i} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{SPS},i}, \mathsf{vk}_{\mathsf{OTS}})$.

Consider now $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct})$:

- By definition, Decrypt sets $\mathsf{ht}_i = 0^\kappa$ for all $i \in [n] \setminus T$.
- By correctness of Π_{SPS}, we have $\mathsf{SPS.Verify}(\mathsf{vk}_{\mathsf{SPS},i}, \mathsf{vk}_{\mathsf{OTS}}, \sigma_{\mathsf{SPS},i}) = 1$ for all $i \in T$. This means
$$C_{\mathsf{ValidSig}}[\mathsf{vk}_{\mathsf{OTS}}](\mathsf{vk}_{\mathsf{SPS},i}, \sigma_{\mathsf{SPS},i}) = 1$$
for all $i \in T$. Since $\beta_i = 0$ for $i \notin T$, we conclude that $\beta_i \le C_{\mathsf{ValidSig}}[\mathsf{vk}_{\mathsf{OTS}}](\mathsf{pk}_i, \mathsf{ht}_i)$ for all $i \in [n]$.
- Since P is monotone, this means
$$1 = P(\beta_1, \ldots, \beta_n) \le P(C_{\mathsf{ValidSig}}[\mathsf{vk}_{\mathsf{OTS}}](\mathsf{pk}_1, \mathsf{ht}_1), \ldots, C_{\mathsf{ValidSig}}[\mathsf{vk}_{\mathsf{OTS}}](\mathsf{pk}_n, \mathsf{ht}_n)).$$
- By correctness of Π_{WE}, the decryption algorithm outputs μ. □

Theorem B.6 (Static Security). *Suppose Π_{OTS} is one-time strongly unforgeable, Π_{SPS} satisfies punctured correctness and punctured key indistinguishability, and Π_{WE} is secure. Then Construction B.4 is statically secure.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the static security game. We begin by defining a sequence of hybrid experiments, each parameterized by a bit $b \in \{0,1\}$:

- $\mathsf{Hyb}_0^{(b)}$: This is the static security experiment with bit $b \in \{0,1\}$:
  - At the beginning of the game, the challenger initializes a counter $\mathsf{ctr} = 0$ and gives $\mathsf{pp} = 1^\lambda$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ can now make queries to the challenger:
    - **Key-generation query:** When $\mathcal{A}$ makes a key-generation query, it must specify a bit $\gamma \in \{0,1\}$. The challenger increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and then samples $(\mathsf{vk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda)$. It sets $\mathsf{pk}_{\mathsf{ctr}} = \mathsf{vk}_{\mathsf{ctr}}$ and responds to $\mathcal{A}$ with $\mathsf{pk}_{\mathsf{ctr}}$.
    - **Corruption query:** On input a counter $\mathsf{ctr}' \le \mathsf{ctr}$, the challenger responds with $\mathsf{sk}_{\mathsf{ctr}}$.
    - **Hint-computation query:** On input a counter $\mathsf{ctr}' \le \mathsf{ctr}$ and a ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, the hint-computation algorithm first checks that $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$. If not, it responds with $\perp$. Otherwise, it responds with $\sigma_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{ctr}'}, \mathsf{vk}_{\mathsf{OTS}})$.
  - In the challenge phase, algorithm $\mathcal{A}$ specifies a policy P and a pair of messages μ_0, μ_1. For each $i \in [n]$, algorithm $\mathcal{A}$ specifies either a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where $\mathcal{A}$ specified a counter value ctr_i, the challenger sets $\mathsf{pk}_i = \mathsf{pk}^*_{\mathsf{ctr}_i}$. The challenger constructs the challenge ciphertext as follows:
    - Sample a key-pair $(\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{sk}^*_{\mathsf{OTS}}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$ for a one-time signature scheme
    - Compute the ciphertext $\mathsf{ct}^*_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$ and the signature $\sigma^*_{\mathsf{OTS}} \leftarrow \mathsf{OTS.Sign}(\mathsf{sk}^*_{\mathsf{OTS}}, \mathsf{ct}^*_{\mathsf{WE}})$

    The challenger replies with the challenge ciphertext $\mathsf{ct}^* = (\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{ct}^*_{\mathsf{WE}}, \sigma^*_{\mathsf{OTS}})$.
  - Algorithm $\mathcal{A}$ can continue to make queries to the challenger. These are handled exactly as in the pre-challenge phase.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$, except the challenger samples the key-pair $(\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{sk}^*_{\mathsf{OTS}}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$ for the challenge ciphertext at the very beginning of the game. Then, when answering hint-computation queries on $(\mathsf{ctr}', \mathsf{ct})$ where $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, the challenger additionally checks the following:
  - If the query is in the pre-challenge phase, the challenger responds with $\perp$ if $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$.
  - If the query is in the post-challenge phase, the challenger responds with $\perp$ if $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$ and $\gamma_{\mathsf{ctr}'} = 0$.

- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$, except when answering key-generation queries where $\gamma = 0$ (i.e., a key for an uncorrupted user), the challenger samples $(\mathsf{vk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{SPS.KeyGenP}(1^\lambda, \mathsf{vk}^*_{\mathsf{OTS}})$.

We write $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ to denote the output distribution of an execution of experiment $\mathsf{Hyb}_i^{(b)}$ with adversary $\mathcal{A}$. We now analyze the hybrid distributions.

Lemma B.7. *If $\mathcal{A}$ is admissible and Π_{OTS} is one-time strongly unforgeable, then for all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Suppose $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| \ge \varepsilon$ for some non-negligible ε. By construction, $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$ are identical experiments unless one of the following two events occur:

- Algorithm $\mathcal{A}$ make a pre-challenge hint-computation query on ctr' and $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$ where $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$ and $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$.
- Algorithm $\mathcal{A}$ makes a post-challenge hint-computation query on ctr' and $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$ where $\gamma_{\mathsf{ctr}'} = 0$, $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$ and $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$.

Thus, in an execution of $\mathsf{Hyb}_0^{(b)}$ or $\mathsf{Hyb}_1^{(b)}$, algorithm $\mathcal{A}$ will trigger one or more of these events with probability at least ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks the one-time unforgeability of Π_{OTS}:

1. At the beginning of the security game, algorithm $\mathcal{B}$ receives a verification key $\mathsf{vk}^*_{\mathsf{OTS}}$.
2. Algorithm $\mathcal{A}$ sets $\mathsf{ctr} = 0$ and $\mathsf{pp} = 1^\lambda$. It starts running $\mathcal{A}$ on input pp. Whenever algorithm $\mathcal{A}$ makes a query, algorithm $\mathcal{B}$ proceeds as follows:
   - **Key-generation query:** When $\mathcal{A}$ makes a key-generation query algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr}+1$. Then it samples $(\mathsf{vk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda)$. It sets $\mathsf{pk}_{\mathsf{ctr}} = \mathsf{vk}_{\mathsf{ctr}}$ and responds to $\mathcal{A}$ with $\mathsf{pk}_{\mathsf{ctr}}$.
   - **Corruption query:** On input a counter $\mathsf{ctr}' \le \mathsf{ctr}$, algorithm $\mathcal{B}$ responds with $\mathsf{sk}_{\mathsf{ctr}}$.
   - **Hint-computation query:** On input a counter $\mathsf{ctr}' \le \mathsf{ctr}$ and a ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, the hint-computation algorithm first checks that $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$. If not, it replies with output $\perp$. Otherwise, if $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$, then algorithm $\mathcal{B}$ halts with output $(\mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$. Otherwise, it responds with $\sigma_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{ctr}'}, \mathsf{vk}_{\mathsf{OTS}})$.
3. During the challenge phase, algorithm $\mathcal{A}$ specifies a policy P and a pair of messages μ_0, μ_1. For each $i \in [n]$, algorithm $\mathcal{A}$ specifies either a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where $\mathcal{A}$ specified a counter value ctr_i, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i = \mathsf{pk}^*_{\mathsf{ctr}_i}$. Algorithm $\mathcal{B}$ then computes
$$\mathsf{ct}^*_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu$$
   Algorithm $\mathcal{B}$ make a signing query on message $\mathsf{ct}_{\mathsf{WE}}$ to its challenger and receives a signature σ^*_{OTS}. It gives the challenge ciphertext $\mathsf{ct}^* = (\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{ct}^*_{\mathsf{WE}}, \sigma^*_{\mathsf{OTS}})$ to $\mathcal{A}$.
4. Algorithm $\mathcal{A}$ can continue making corruption and hint-computation queries. Algorithm $\mathcal{B}$ responds to corruption queries using the same procedure described above. For a hint-computation query on a counter $\mathsf{ctr}' \le \mathsf{ctr}$ and a ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, algorithm $\mathcal{B}$ first checks that $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$. If not, it responds with $\perp$. Otherwise, if $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$ and $\gamma_{\mathsf{ctr}'} = 0$, then algorithm $\mathcal{B}$ halts with output $\mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$. Otherwise, it responds with $\sigma_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{ctr}'}, \mathsf{vk}_{\mathsf{OTS}})$. It responds to $\mathcal{A}$ with σ_{SPS}.

By construction, the challenger samples $(\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{sk}^*_{\mathsf{OTS}}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$ which matches the specification in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$. Next, the key-generation and corruption queries are handled using the same procedure as in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$. Moreover, in the challenge phase, the challenger would compute $\sigma^*_{\mathsf{OTS}} \leftarrow \mathsf{OTS.KeyGen}(\mathsf{sk}^*_{\mathsf{OTS}}, \mathsf{ct}^*_{\mathsf{WE}})$, which again matches the distribution in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$. We conclude that $\mathcal{B}$ perfectly simulates the challenger's behavior in $\mathsf{Hyb}_0^{(b)}$, so with probability ε, one of the two events above will occur with probability as least ε:

- Suppose $\mathcal{A}$ make a pre-challenge hint-computation query on ctr' and $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$ where we have $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$ and $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$. Since this is a *pre-challenge* query, algorithm $\mathcal{B}$ has not made any signing queries to its oracle yet. This means $(\mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$ is a valid forgery for $\mathsf{vk}^*_{\mathsf{OTS}}$.
- Suppose $\mathcal{A}$ makes a post-challenge hint-computation query on ctr' and $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$ where $\gamma_{\mathsf{ctr}'} = 0$, $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$ and $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$. Since $\mathcal{A}$ is admissible for the static security game, $\gamma_{\mathsf{ctr}'} = 0$, and this is a post-challenge query, it must be the case that $\mathsf{ct} \neq \mathsf{ct}^*$. This means either $\mathsf{ct}_{\mathsf{WE}} \neq \mathsf{ct}^*_{\mathsf{WE}}$ or $\sigma_{\mathsf{OTS}} \neq \sigma^*_{\mathsf{OTS}}$. Correspondingly $(\mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$ is a valid forgery for $\mathsf{vk}^*_{\mathsf{OTS}}$. Since there is the possibility that $\mathsf{ct}_{\mathsf{WE}} = \mathsf{ct}^*_{\mathsf{WE}}$ (but $\sigma_{\mathsf{OTS}} \neq \sigma^*_{\mathsf{OTS}}$), we rely on strong unforgeability here.

Thus, we conclude that if either of the two events defined above occurs, algorithm $\mathcal{B}$ successfully breaks one-time strong unforgeability of Π_{OTS}. The claim follows. □

Lemma B.8. *If $\mathcal{A}$ is admissible and Π_{SPS} satisfies punctured key indistinguishability, then for all $b \in \{0,1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Let Q be a bound on the number of key-generation queries algorithm $\mathcal{A}$ makes. We now define an intermediate sequence of hybrid experiments indexed by $i \in [0, Q]$:

- $\mathsf{Hyb}_{1,i}^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$, except the challenger responds to the first i key-generation queries that algorithm $\mathcal{A}$ makes using the specification in $\mathsf{Hyb}_2^{(b)}$.

We now show that for all $i \in [Q]$, $\mathsf{Hyb}_{1,i-1}^{(b)}$ and $\mathsf{Hyb}_i^{(b)}$ are computationally indistinguishable. Suppose for some index $i \in [Q]$, we have

$$|\Pr[\mathsf{Hyb}_{1,i-1}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,i}^{(b)}(\mathcal{A}) = 1]| \geq \varepsilon$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the punctured key indistinguishability game:

1. Algorithm $\mathcal{B}$ initializes a counter $\mathsf{ctr} = 0$ and samples $(\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{sk}^*_{\mathsf{OTS}}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$. It gives $\mathsf{vk}^*_{\mathsf{OTS}}$ to the challenger and receives a key $\mathsf{vk}^*_{\mathsf{SPS}}$.
2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ with input $\mathsf{pp} = 1^\lambda$. Whenever algorithm $\mathcal{A}$ makes a query, algorithm $\mathcal{B}$ proceeds as follows:
   - **Key-generation query:** When $\mathcal{A}$ makes a key-generation query with corruption bit $\gamma \in \{0,1\}$, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$. Then it proceeds as follows:
     - Suppose $\gamma = 1$. Then it samples $(\mathsf{vk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda)$.
     - Suppose $\gamma = 0$. If $\mathsf{ctr} < i$, then it samples $(\mathsf{vk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{SPS.KeyGenP}(1^\lambda, \mathsf{vk}^*_{\mathsf{OTS}})$. If $\gamma > i$, then it samples $(\mathsf{vk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda)$. If $\gamma = i$, then it sets $\mathsf{vk}_{\mathsf{ctr}} = \mathsf{vk}^*_{\mathsf{SPS}}$.

     Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with $\mathsf{vk}_{\mathsf{ctr}}$.
   - **Corruption query:** If $\mathcal{A}$ makes a corruption query on an index $\mathsf{ctr}' \leq \mathsf{ctr}$, algorithm $\mathcal{B}$ aborts with output 0 if $\gamma_{\mathsf{ctr}'} = 0$. Otherwise, it responds with $\mathsf{sk}_{\mathsf{ctr}'}$.
   - **Hint-computation query:** If $\mathcal{A}$ makes a hint-computation query on a counter $\mathsf{ctr}' \leq \mathsf{ctr}$ and a ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, algorithm $\mathcal{B}$ first checks that $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$. If not, it responds with $\perp$. Otherwise, it proceeds as follows:
     - If $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$, algorithm $\mathcal{B}$ responds with $\perp$.
     - Otherwise, if $\mathsf{ctr}' \neq i$ or if $\gamma_{\mathsf{ctr}'} = 1$, then algorithm $\mathcal{B}$ responds with $\sigma_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{ctr}'}, \mathsf{vk}_{\mathsf{OTS}})$.
     - Finally, if $\mathsf{ctr}' = i$ and $\gamma_{\mathsf{ctr}'} = 0$, algorithm $\mathcal{B}$ makes a signing query on the message $\mathsf{vk}_{\mathsf{OTS}}$ to receive a signature σ_{SPS}. It responds to $\mathcal{A}$ with σ_{SPS}.

3. During the challenge phase, algorithm $\mathcal{A}$ specifies a policy P and a pair of messages μ_0, μ_1. For each $i \in [n]$, algorithm $\mathcal{A}$ specifies either a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where $\mathcal{A}$ specified a counter value ctr_i, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i = \mathsf{pk}^*_{\mathsf{ctr}_i}$. Algorithm $\mathcal{B}$ then computes

$$\begin{aligned}\mathsf{ct}^*_{\mathsf{WE}} &\leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)\\ \sigma^*_{\mathsf{OTS}} &\leftarrow \mathsf{OTS.Sign}(\mathsf{sk}^*_{\mathsf{OTS}}, \mathsf{ct}^*_{\mathsf{WE}}).\end{aligned}$$

Algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct}^* = (\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{ct}^*_{\mathsf{WE}}, \sigma^*_{\mathsf{OTS}})$.

4. Algorithm $\mathcal{A}$ can continue making corruption and hint-computation queries. Algorithm $\mathcal{B}$ responds to corruption queries using the same procedure described above. For a hint-computation query on a counter $\mathsf{ctr}' \le \mathsf{ctr}$ and a ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, algorithm $\mathcal{B}$ first checks that $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$. If not, it responds with $\perp$. Otherwise, it proceeds as follows:
   - If $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$ and $\gamma_{\mathsf{ctr}'} = 0$, then algorithm $\mathcal{B}$ responds with $\perp$.
   - Otherwise, if $\mathsf{ctr}' \neq i$ or if $\gamma_{\mathsf{ctr}'} = 1$, then algorithm $\mathcal{B}$ responds with $\sigma_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{ctr}'}, \mathsf{vk}_{\mathsf{OTS}})$.
   - Finally, if $\mathsf{ctr}' = i$ and $\gamma_{\mathsf{ctr}'} = 0$, then algorithm $\mathcal{B}$ makes a signing query on the message $\mathsf{vk}_{\mathsf{OTS}}$ to receive a signature σ_{SPS}. It responds to $\mathcal{A}$ with σ_{SPS}.

5. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{A}$ also outputs.

By construction, algorithm $\mathcal{B}$ does not make any signing queries on $\mathsf{vk}^*_{\mathsf{OTS}}$, so it is a valid adversary for the punctured key indistinguishability game. Next, algorithm $\mathcal{B}$ responds to corruption queries on counters ctr where $\gamma_{\mathsf{ctr}} = 0$ with $\perp$ (rather than the associated secret key). However, when $\mathcal{A}$ is admissible, it is not allowed to issue corruption queries on ctr where $\gamma_{\mathsf{ctr}} = 0$. Thus, as long as $\mathcal{A}$ is admissible, the behavior of $\mathcal{B}$ perfectly coincides with the specification in $\mathsf{Hyb}^{(b)}_{1,i-1}$ and $\mathsf{Hyb}^{(b)}_{1,i}$. Now, if the challenger samples $(\mathsf{vk}^*_{\mathsf{SPS}}, \mathsf{sk}^*_{\mathsf{SPS}}) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda)$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}^{(b)}_{1,i-1}$ whereas if the challenger samples $(\mathsf{vk}^*_{\mathsf{SPS}}, \mathsf{sk}^*_{\mathsf{SPS}}) \leftarrow \mathsf{SPS.KeyGenP}(1^\lambda, \mathsf{vk}^*_{\mathsf{OTS}})$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}^{(b)}_{1,i}$. We conclude that $\mathcal{B}$ breaks punctured key indistinguishability with the same advantage. Finally, algorithm $\mathcal{B}$ is efficient so $Q = \mathsf{poly}(\lambda)$. Lemma B.8 now follows by a hybrid argument. □

Lemma B.9. *If $\mathcal{A}$ is admissible, Π_{SPS} satisfies punctured correctness, and Π_{WE} is secure, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(0)}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(1)}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Suppose $|\Pr[\mathsf{Hyb}^{(0)}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(1)}_2(\mathcal{A}) = 1]| \ge \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the witness encryption security game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ initializes a counter $\mathsf{ctr} = 0$ and samples $(\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{sk}^*_{\mathsf{OTS}}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$. It gives $\mathsf{vk}^*_{\mathsf{OTS}}$ to the challenger and receives a key $\mathsf{vk}^*_{\mathsf{SPS}}$.

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ with input $\mathsf{pp} = 1^\lambda$. Whenever algorithm $\mathcal{A}$ makes a query, algorithm $\mathcal{B}$ proceeds as follows:
   - **Key-generation query:** When $\mathcal{A}$ makes a key-generation query with corruption bit $\gamma \in \{0,1\}$, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$. Then it proceeds as follows:
     - If $\gamma = 0$, it samples $(\mathsf{vk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{SPS.KeyGenP}(1^\lambda, \mathsf{vk}^*_{\mathsf{OTS}})$.
     - If $\gamma = 1$, it samples $(\mathsf{vk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda)$.

     Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with $\mathsf{vk}_{\mathsf{ctr}}$.
   - **Corruption query:** If $\mathcal{A}$ makes a corruption query on an index $\mathsf{ctr}' \le \mathsf{ctr}$, algorithm $\mathcal{B}$ responds with $\mathsf{sk}_{\mathsf{ctr}'}$.

- **Hint-computation query:** If $\mathcal{A}$ makes a hint-computation query on a counter $\mathsf{ctr}' \le \mathsf{ctr}$ and a ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, algorithm $\mathcal{B}$ first checks that $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$. If not, it responds with $\perp$. If $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$, it also responds with $\perp$. Otherwise, it responds with $\sigma_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{ctr}'}, \mathsf{vk}_{\mathsf{OTS}})$.

3. During the challenge phase, algorithm $\mathcal{A}$ specifies a policy P and a pair of messages μ_0, μ_1. For each $i \in [n]$, algorithm $\mathcal{A}$ specifies either a public key pk_i or a counter value ctr_i. For each $i \in [n]$ where $\mathcal{A}$ specified a counter value ctr_i, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i = \mathsf{pk}^*_{\mathsf{ctr}_i}$. Algorithm $\mathcal{B}$ gives the circuit $C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}]$, the policy P, the instances $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and the pair of messages μ_0, μ_1 to the challenger. The challenger replies with $\mathsf{ct}^*_{\mathsf{WE}}$. Finally, algorithm $\mathcal{B}$ computes $\sigma^*_{\mathsf{OTS}} \leftarrow \mathsf{OTS.Sign}(\mathsf{sk}^*_{\mathsf{OTS}}, \mathsf{ct}^*_{\mathsf{WE}})$, and responds to $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct}^* = (\mathsf{vk}^*_{\mathsf{OTS}}, \mathsf{ct}^*_{\mathsf{WE}}, \sigma^*_{\mathsf{OTS}})$.

4. Algorithm $\mathcal{A}$ can continue making corruption and hint-computation queries. Algorithm $\mathcal{B}$ responds to corruption queries using the same procedure described above. For a hint-computation query on a counter $\mathsf{ctr}' \le \mathsf{ctr}$ and a ciphertext $\mathsf{ct} = (\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}})$, algorithm $\mathcal{B}$ first checks that $\mathsf{OTS.Verify}(\mathsf{vk}_{\mathsf{OTS}}, \mathsf{ct}_{\mathsf{WE}}, \sigma_{\mathsf{OTS}}) = 1$. If $\gamma_{\mathsf{ctr}'} = 0$ and $\mathsf{vk}_{\mathsf{OTS}} = \mathsf{vk}^*_{\mathsf{OTS}}$, then it also responds with $\perp$. Otherwise, it responds with $\sigma_{\mathsf{SPS}} \leftarrow \mathsf{SPS.Sign}(\mathsf{sk}_{\mathsf{ctr}'}, \mathsf{vk}_{\mathsf{OTS}})$.

5. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{A}$ also outputs.

First, we argue that for all $\sigma_1, \ldots, \sigma_n \in \{0,1\}^\kappa$,

$$P\left(C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}](\mathsf{pk}_1, \sigma_1), \ldots, C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}](\mathsf{pk}_n, \sigma_n)\right) = 0. \tag{B.1}$$

Let $\gamma_1, \ldots, \gamma_{\mathsf{ctr}}$ be the corruption bits associated with the adversary's key-generation queries. We proceed as follows:

- First, for all $j \in [\mathsf{ctr}]$ where $\gamma_j = 0$, there does not exist $\sigma \in \{0,1\}^\kappa$ where $C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}](\mathsf{pk}_j, \sigma) = 1$. By definition, $C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}](\mathsf{pk}_j, \sigma) = 1$ only if $\mathsf{SPS.Verify}(\mathsf{pk}_j, \mathsf{vk}^*_{\mathsf{OTS}}, \sigma) = 1$. However, when $\gamma_j = 0$, algorithm $\mathcal{B}$ samples $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{SPS.KeyGen}(1^\lambda, \mathsf{vk}^*_{\mathsf{OTS}})$. By punctured correctness of Π_{SPS}, with overwhelming probability over the choice of $(\mathsf{pk}_j, \mathsf{sk}_j)$, there does not exist σ where $\mathsf{SPS.Verify}(\mathsf{pk}_j, \mathsf{vk}^*_{\mathsf{OTS}}, \sigma) = 1$. The claim now holds by a union bound over all of the key-generation queries $j \in [\mathsf{ctr}]$ where $\gamma_j = 0$.

- For each $i \in [n]$, let $\beta_i = 1$ if the adversary specified the public key pk_i in the challenge phase or if it chose a counter value ctr_i where $\gamma_{\mathsf{ctr}_i} = 1$. Let $\beta_i = 0$ otherwise. By admissibility, we have that $P(\beta_1, \ldots, \beta_n) = 0$.

- Take any candidate witness $(\sigma_1, \ldots, \sigma_n)$. Let $\beta'_i = C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}](\mathsf{pk}_i, \sigma_i)$. From the first property, on all indices $i \in [n]$ where algorithm $\mathcal{A}$ specified a counter value ctr_i where $\gamma_{\mathsf{ctr}_i} = 0$, with overwhelming probability, $\beta'_i = C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}](\mathsf{pk}_i, \sigma_i) = 0 = \beta_i$. On indices $i \in [n]$ where algorithm $\mathcal{A}$ chosen the public key pk_i or a counter value ctr_i where $\gamma_{\mathsf{ctr}_i} = 1$, we have $\beta_i = 1$. We conclude that for all $i \in [n]$, $\beta'_i \le \beta_i$. Since P is monotone, this means
$$P(\beta'_1, \ldots, \beta'_n) \le P(\beta_1, \ldots, \beta_n) = 0,$$
as required.

Thus, for all $\sigma_1, \ldots, \sigma_n \in \{0,1\}^\lambda$, Eq. (B.1) holds. In this case, the witness encryption challenger encrypts either μ_0 or μ_1. If $\mathsf{ct}^*_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_0)$, then algorithm $\mathcal{B}$ simulates an execution of $\mathsf{Hyb}_2^{(0)}$ with overwhelming probability. Conversely, if $\mathsf{ct}^*_{\mathsf{WE}} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, C_{\mathsf{ValidSig}}[\mathsf{vk}^*_{\mathsf{OTS}}], P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu_1)$, then simulates an execution of $\mathsf{Hyb}_2^{(1)}$ with overwhelming probability. Algorithm $\mathcal{B}$ breaks witness encryption with the same advantage ε. □

Static security now follows by combining Lemmas B.7 to B.9. □