Constrained Pseudorandom Functions for Unconstrained Inputs

Apoorvaa Deshpande
acdeshpa@cs.brown.edu *

Venkata Koppula
kvenkata@cs.utexas.edu

Brent Waters
bwaters@cs.utexas.edu†

Abstract

A constrained pseudo random function (PRF) behaves like a standard PRF, but with the added feature that the (master) secret key holder, having secret key K, can produce a constrained key, $K\{f\}$, that allows for the evaluation of the PRF on all inputs satisfied by the constraint f. Most existing constrained PRF constructions can handle only bounded length inputs. In a recent work, Abusalah et al. [AFP14] constructed a constrained PRF scheme where constraints can be represented as Turing machines with unbounded inputs. Their proof of security, however, requires risky "knowledge type" assumptions such as (public coins) differing inputs obfuscation for circuits and SNARKs.

In this work, we construct a constrained PRF scheme for Turing machines with unbounded inputs under weaker assumptions, namely, the existence of indistinguishability obfuscation for circuits (and DDH).

*This work was done while the author was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

†Supported by NSF CNS-0952692, CNS-1228599 and CNS-1414082. DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

1 Introduction

Constrained pseudorandom functions (PRFs), as introduced by [BW13, BGI14, KPTZ13], are a useful extension of standard PRFs [GGM84]. A constrained PRF system is defined with respect to a *family of constraint functions*, and has an additional algorithm Constrain. This algorithm allows a (master) PRF key holder, having PRF key K, to produce a *constrained* PRF key $K\{f\}$ corresponding to a constraint f. This constrained key $K\{f\}$ can be used to evaluate the PRF at all points x accepted by f (that is, $f(x) = 1$). The security notion ensures that even when given multiple constrained keys $K\{f_1\}$, ..., $K\{f_Q\}$, PRF evaluation at a point not accepted by any of the functions f_i 'looks' uniformly random to a computationally bounded adversary. Since their inception, constrained PRFs have found several applications such as broadcast encryption, identity-based key exchange, policy-based key distribution[BW13] and multi-party key exchange [BZ14]. In particular, even the most basic class of constrained PRFs called puncturable PRFs has found immense application in the area of program obfuscation through the 'punctured programming' technique introduced by [SW14]. The initial works of [BW13, BGI14, KPTZ13] showed that the [GGM84] PRF construction can be modified to construct a basic class of constrained PRFs called prefix-constrained PRFs (which also includes puncturable PRFs). Boneh and Waters [BW13] also showed a construction for the richer class of circuit-constrained PRFs [1] using multilinear maps [GGH13a]. Since then, we have seen great progress in this area, leading to constructions from different cryptographic assumptions [BZ14, BV15b, BFP+15] and constructions with additional properties [CRV14, AFP14, BV15b, BFP+15]. However, all the above mentioned works have a common limitation: the corresponding PRF can handle only bounded length inputs.

The problem of constructing constrained PRFs with unbounded length was studied in a recent work by Abusalah, Fuchsbauer and Pietrzak [AFP14], who also showed motivating applications such as broadcast encryption with unbounded recipients and multi-party identity based non-interactive key exchange with no apriori bound on number of parties. Abusalah et al. construct a constrained PRF scheme where the constraint functions are represented as Turing machines with unbounded inputs. Initially, their scheme was proven secure under the assumption that differing input obfuscation ($di\mathcal{O}$) for circuits exists. Informally, this assumption states that there exists an 'obfuscation' program $\mathcal{O}$ that takes as input a circuit C, and outputs another circuit $\mathcal{O}(C)$ with the following security guarantee: if an efficient adversary can distinguish between $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$, then there exists an efficient extraction algorithm that can find an input x such that $C_1(x) \neq C_2(x)$. However, the $di\mathcal{O}$ assumption is believed to be a risky one due to its 'extractability nature'. Furthermore, the work of [GGHW14] conjectures that there exist certain function classes for which $di\mathcal{O}$ is impossible to achieve. Subsequently, they showed a construction [AFP16] under the assumption of public coins differing input obfuscation ($pcdi\mathcal{O}$) [IPS15]. While the implausibility results of [GGHW14] does not apply to $pcdi\mathcal{O}$, it is still believed to be a strong assumption due to its extractability nature.

A natural direction then is to try to base the security on the relatively weaker assumption of indistinguishability obfuscation ($i\mathcal{O}$) for circuits. An obfuscator $\mathcal{O}$ is an indistinguishability obfuscator for circuits if for any two circuits C_1 and C_2 that have identical functionality, their obfuscations $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ are computationally indistinguishable. Unlike $di\mathcal{O}$, there are no known impossibility results for $i\mathcal{O}$, and moreover, there has been recent progress [GLSW15, AJ15, BV15a] towards the goal of constructing $i\mathcal{O}$ from standard assumptions. This brings us to the central question of our work:

Can we construct constrained PRFs for Turing machines under the assumptions that indistinguishability obfuscation and one-way functions exist?

Our starting point is three recent works that build indistinguishability obfuscation for Turing Machines with bounded length inputs using $i\mathcal{O}$ for circuits [BGL+15, CHJV15, KLW15]. The works of [BGL+15, CHJV15] show how to do this where the encoding time and size of the obfuscated program grows with the maximum space used by the underlying program, whereas the work of [KLW15] achieves this with no such restriction. An immediate question is whether we can use a Turing machine obfuscator for constructing constrained PRFs for Turing machines, similar to the circuit-constrained PRF construction of [BZ14]. However, as mentioned above the Turing machine obfuscator constructions are restricted to Turing

[1] Where the constraints can be any boolean circuit

Machines with bounded size inputs[2]. Thus, we are unable to use the Turning Machine obfuscation scheme in a black box manner and have to introduce new techniques to construct constrained PRFs for unbounded sized inputs.

Our Results: The main result of our work is as follows.

Theorem 1 (informal). *Assuming the existence of secure indistinguishability obfuscators and DDH, there exists a constrained PRF scheme that is selectively secure.*

Selective Security vs Adaptive Security: Selective security is a security notion where the adversary must specify the 'challenge input' before receiving constrained keys. A stronger notion, called adaptive security, allows the adversary to query for constrained keys before choosing the challenge input. While adaptive security should be the ideal target, achieving adaptive security with only polynomial factor security loss (i.e. without 'complexity leveraging') has been challenging, even for circuit based constrained PRFs. Currently, the best known results for adaptive security either require superpolynomial security loss [FKPR14], or work for very restricted functionalities [HKW15], or achieve non-collusion based security [BV15b] or achieve it in the random oracle mode [HKKW14].

Moreover, for many applications, it turns out that selective security is sufficient. For example, the widely used punctured programming technique of [SW14] only requires selectively secure puncturable PRFs. Similarly, as discussed in [AFP14], selectively secure constrained PRFs with unbounded inputs can be used to construct broadcast encryption schemes with unbounded recipients and identity based non-interactive key exchange (ID-NIKE) protocol with no apriori bound on number of parties. Therefore, as a corollary of Theorem 1, we get both these applications using only indistinguishability obfuscation and DDH. Interestingly, two recent works have shown direct constructions for both these problems using $i\mathcal{O}$. Zhandry [Zha14] showed a broadcast encryption scheme with unbounded recipients, while Khurana et al. [KRS15] showed an ID-NIKE scheme with unbounded number of parties.

We also show how our construction above can be easily adapted to get selectively secure attribute based encryption for Turing machines with unbounded inputs, which illustrates the versatility of our techniques above.

Theorem 2 (informal). *Assuming the existence of secure indistinguishability obfuscators and DDH, there exists an ABE scheme for Turing machines that is selectively secure.*

Recently, Ananth and Sahai [AS16] had an exciting result where they show adaptively secure functional encryption for Turing machines with unbounded inputs. While our adaptation is limited to ABE, we believe that the relative simplicity of our construction is an interesting feature. In addition, we were able to apply our tail-hybrid approach to get an end-to-end polynomial time reduction.

1.1 Overview of our constrained PRF construction

To begin, let us consider the simple case of standard PRFs with unbounded inputs. Any PRF (with sufficient input size) can be extended to handle unbounded inputs by first compressing the input using a collision-resistant hash function (CRHF), and then computing the PRF on this hash value. Abusalah et al. [AFP14] showed that by using $di\mathcal{O}$, this approach can be extended to work for constrained PRFs. However, the proof of security relies on the extractability property of $di\mathcal{O}$ in a fundamental way. In particular, this approach will not work if $i\mathcal{O}$ is used instead of $di\mathcal{O}$ because general CRHFs are not '$i\mathcal{O}$-compatible'[3] (see Section 2 for a more detailed discussion on $i\mathcal{O}$-compatibility).

[2]The restriction to bounded length inputs is due to the fact that their iO analysis requires a hybrid over all possible inputs. They absorb this loss by growing the size of the obfuscated program polynomially in the input size using complexity leveraging and a sub-exponential hardness assumption on the underlying circuit iO. Currently, there is no known way to avoid this.

[3]Consider the following toy example. Let C_0, C_1 be circuits such that $C_0(x, y) = 0\ \forall (x, y)$ and $C_1(x, y) = 1$ iff $\mathsf{CRHF}(x) = \mathsf{CRHF}(y)$ for $x \neq y$. Now, under the $di\mathcal{O}$ assumption, the obfuscations of C_0 and C_1 are computationally indistinguishable. However, we cannot get the same guarantee by using $i\mathcal{O}$, since the circuits are not functionally identical

Challenges of a similar nature were addressed in [KLW15] by introducing new tools and techniques that guarantee program functional equivalence at different stages of the proof. Let us review one such tool called *positional accumulators*, and see why it is $i\mathcal{O}$-compatible. A positional accumulator scheme is a cryptographic primitive used to provide a short commitment to a much larger storage. This commitment (also referred to as an accumulation of the storage) has two main features: succinct verifiability (there exists a short proof to prove that an element is present at a particular position) and succinct updatability (using short auxiliary information, the accumulation can be updated to reflect an update to the underlying storage). The scheme also has a setup algorithm which generates the parameters, and can operate in two computationally indistinguishable modes. It can either generate parameters 'normally', or it can be *enforcing* at a particular position p. When parameters are generated in the enforcing mode, the accumulator is *information-theoretically binding* to position p of the underlying storage. This information theoretic enforcing property is what makes it compatible for proofs involving $i\mathcal{O}$. The original definition of positional accumulators required these enforcing modes to take the entire 'history' up to step i as input; however the recent works of Ananth et al [ACC+15] and Canetti et al [CCHR15] construct positional accumulators which only require the positional p. This variant, referred to as history-less positional accumulator [ACC+15] or adaptive positional accumulators [CCHR15], will be essential for our proof.

Returning to our constrained PRF problem, we need a special hash function that can be used with $i\mathcal{O}$. That brings us to the main insight of our work: the history-less/adaptive positional accumulator can be repurposed to be an $i\mathcal{O}$-friendly hash function.[4] Besides giving us an $i\mathcal{O}$-friendly hash function, this also puts the input in a data structure that is already suitable for the KLW framework.[5]

Our Construction : We will now sketch out our construction. Our constrained PRF scheme uses a puncturable PRF F with key k. Let $\mathsf{Hash\text{-}Acc}(x)$ represent the accumulation of storage initialized with input $x = x_1 \dots x_n$. The PRF evaluation (in our scheme) is simply $F(k, \mathsf{Hash\text{-}Acc}(x))$.

The interesting part is the description of our constrained keys, and how they can be used to evaluate at an input x. The constrained key for machine M consists of two programs. The first one is an obfuscated circuit which takes an input, and outputs a signature on that input. The second one is an obfuscated circuit which essentially computes the next-step of the Turing machine, and eventually, if it reaches the 'accepting state', it outputs $F(k, \mathsf{Hash\text{-}Acc}(x))$. This circuit also performs additional authenticity checks to prevent illegal inputs - it takes a signature and accumulator as input, verifies the signature and accumulator before computing the next step, and finally updates the accumulator and outputs a signature on the new state and accumulator.

Evaluating the PRF at input x using the constrained key consists of two steps. The first one is the initialization step, where the evaluator first computes $\mathsf{Hash\text{-}Acc}(x)$ and then computes a signature on $\mathsf{Hash\text{-}Acc}(x)$ using the signing program. Then, it iteratively runs the obfuscated next-step circuit (also including $\mathsf{Hash\text{-}Acc}(x)$ as input at each time step) until the circuit either outputs the PRF evaluation, or outputs $\perp$. While this is similar to the KLW message hiding encoding scheme, there are some major differences. One such difference is with regard to accumulation of the input. In KLW, the input is accumulated by the 'honest' encoding party, while in our case, the (possibly corrupt) evaluator generates the accumulation and feeds it at each step of the iteration. As a result, the KLW proof for message-hiding encoding scheme needs to be tailored to fit our setting. Secondly, in the case of constrained PRFs, we need to generate the parameters before receiving constrained key queries. As a result, this requires the enforcing to be adaptive.

Proof of Security : Recall we are interested in proving selective security, where the adversary sends the challenge input x^* before requesting for constrained keys. Our goal is to replace the (master) PRF key k in all constrained keys with one that is punctured at $\mathsf{acc\text{-}inp}^* = \mathsf{Hash\text{-}Acc}(x^*)$. Once this is done, the security of puncturable PRFs guarantees that the adversary cannot distinguish between $\mathsf{F}(k, \mathsf{acc\text{-}inp}^*)$ and a truly

[4] More formally, it gives us an $i\mathcal{O}$ friendly universal one way hash function.

[5] We note that the *somewhat statistically binding* hash of [HW15] has a similar spirit to positional accumulators in that they have statistical binding at a selected position. However, they are not sufficient for our purposes as positional accumulators provide richer semantics such as interleaved reads, writes, and overwrites that are necessary here.

random string. Let us focus our attention on one constrained key query corresponding to machine M, and suppose M runs for t^* steps on input x^* and finally outputs 'reject'.

To replace k with a punctured key, we need to ensure that the obfuscated program for M does not reach the 'accepting state' on inputs with $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. This is done via two main hybrid steps. First, we alter the program so that it does not reach the accepting state within t^* steps on inputs with $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. Then, we have the *tail hybrid*, where we ensure that on inputs with $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, the program does not reach accepting state even at time steps $t > t^*$. For the first step, we follow the KLW approach together with the history-less/adaptive accumulators, and define a sequence of t^* sub-hybrids, where in the i^{th} hybrid, the obfuscated circuit does not reach accepting state at time steps $t \leq i$ for inputs with $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. We use the KLW selective enforcement techniques to show that consecutive hybrids are computationally indistinguishable.

We have a novel approach for handling the tail hybrids Let $T(= 2^\lambda)$ denote the upper bound on the running time of any machine M on any input. In KLW, the tail hybrid step was handled by defining $T - t^*$ intermediate hybrids. If we adopt a similar approach for our construction, it results in an exponential factor security loss, which is undesirable for our application[6]. Our goal would be to overcome this to get an end to end polynomial reduction to $i\mathcal{O}$. Therefore, we propose a modification to our scheme which will allow us to handle the tail hybrid with only a polynomial factor security loss. First, let us call the time step 2^i as the i^{th} *landmark*, while the interval $[2^i, 2^{i+1} - 1]$ is the i^{th} interval. The obfuscated program now takes a PRG seed as input at each time step, and performs some additional checks on the input PRG seed. At time steps just before a landmark, it outputs a new (pseudorandomly generated) PRG seed, which is then used in the next interval. Using standard $i\mathcal{O}$ techniques, we can show that if the program outputs $\perp$ just before a landmark, then we can alter the program indistinguishably so that it outputs $\perp$ at all time steps in the next interval. Since we know that the program outputs $\perp$ at $(\mathsf{acc\text{-}inp}^*, t^* - 1)$, we can ensure that the program outputs $\perp$ for all $(\mathsf{acc\text{-}inp}^*, t)$ such that $t^* \leq t \leq 2t^*$. Proceeding inductively, we can ensure that the program never reaches accepting state if $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$.

Concurrent Work Independently and concurrently, Abusalah and Fuchsbauer [AF16] showed how to improve the [AFP16] scheme to construct constrained PRFs for unbounded inputs with short keys. However, the security of this scheme also relies on the public coin $di\mathcal{O}$ assumption.

Errata The previous version of this paper used the KLW definition of positional accumulators. In this definition, as mentioned above, the enforcing mode setup requires the entire history up to step i. However, in the selective definition, the challenger must output the PRF evaluation before receiving the constrained key queries. Since the accumulator setup needs to be performed before the PRF evaluation, this results in an error in our proof. This issue was pointed out in the recent work of [DDM16]. In this version, we rectify it by using history-less positional accumulators instead of the original KLW definition. Once this is done, our proof goes through in a similar pattern as before. Ananth et al showed how to construct history-less positional accumulators from DDH and ϕ-assumption.

An alternate fix would be to consider a weaker security definition for constrained PRFs, where the queries are also specified before receiving the challenge. The original definition of positional accumulators suffices for this weaker security definition.

1.2 Attribute Based Encryption for Turing Machines with Unbounded Inputs

We will now describe our ABE scheme for Turing machines with unbounded inputs. Let $\mathcal{PKE}$ be a public key encryption scheme. Our ABE scheme's master secret key is a puncturable PRF key k and the public key is an obfuscated program $\mathsf{Prog\text{-}PK}$ and accumulator parameters. The program $\mathsf{Prog\text{-}PK}$ takes as input a string $\mathsf{acc\text{-}inp}$, computes $r = F(k, \mathsf{acc\text{-}inp})$ and uses r as randomness for $\mathsf{PKE.setup}$. It finally outputs the $\mathcal{PKE}$ public key. To encrypt a message m for attribute x, one must first accumulate the input x, then feed

[6] An exponential loss in the security proof of randomized encodings in KLW was acceptable because the end goal was indistinguishability obfuscation, which already requires an exponential number of hybrids.

the accumulated input to Prog-PK to get a $\mathcal{PKE}$ public key pk, and finally encrypts m using public key pk. The secret keys corresponding to Turing machine M is simply the constrained PRF key for M. This key can be used to compute $F(k, \mathsf{Hash\text{-}Acc}(x))$ if $M(x) = 1$, and therefore can decrypt messages encrypted for x.

1.3 Paper Organization

We present the required preliminaries in Section 2 and the notions of constrained PRFs for Turing machines in Section 3. The construction of our constrained PRF scheme can be found in Section 4, while our ABE scheme can be found in 5.

2 Preliminaries

2.1 Notations

In this work, we will use the following notations for Turing machines.

Turing machines A Turing machine is a 7-tuple $M = \langle Q, \Sigma_{\text{tape}}, \Sigma_{\text{inp}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$ with the following semantics:

- Q is the set of states with start state q_0, accept state q_{ac} and reject state q_{rej}.
- Σ_{inp} is the set of inputs symbols
- Σ_{tape} is the set of tape symbols. We will assume $\Sigma_{\text{inp}} \subset \Sigma_{\text{tape}}$ and there is a special blank symbol '$\sqcup$' $\in \Sigma_{\text{tape}} \setminus \Sigma_{\text{inp}}$.
- $\delta : Q \times \Sigma_{\text{tape}} \to Q \times \Sigma_{\text{tape}} \times \{+1, -1\}$ is the transition function.

2.2 Obfuscation

We recall the definition of indistinguishability obfuscation from [GGH+13b, SW14].

Definition 2.1. (Indistinguishability Obfuscation) Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of polynomial-size circuits. Let $i\mathcal{O}$ be a uniform PPT algorithm that takes as input the security parameter λ, a circuit $C \in \mathcal{C}_\lambda$ and outputs a circuit C'. $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}$ if it satisfies the following conditions:

- (Preserving Functionality) For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that $C'(x) = C(x)$ where $C' \leftarrow i\mathcal{O}(1^\lambda, C)$.
- (Indistinguishability of Obfuscation) For any (not necessarily uniform) PPT distinguisher $\mathcal{B} = (Samp, \mathcal{D})$, there exists a negligible function $\text{negl}(\cdot)$ such that the following holds: if for all security parameters $\lambda \in \mathbb{N}, \Pr[\forall x, C_0(x) = C_1(x) : (C_0; C_1; \sigma) \leftarrow Samp(1^\lambda)] > 1 - \text{negl}(\lambda)$, then

$$\begin{aligned} &|\Pr[\mathcal{D}(\sigma, i\mathcal{O}(1^\lambda, C_0)) = 1 : (C_0; C_1; \sigma) \leftarrow Samp(1^\lambda)] - \\ &\Pr[\mathcal{D}(\sigma, i\mathcal{O}(1^\lambda, C_1)) = 1 : (C_0; C_1; \sigma) \leftarrow Samp(1^\lambda)]| \leq \text{negl}(\lambda). \end{aligned}$$

In a recent work, [GGH+13b] showed how indistinguishability obfuscators can be constructed for the circuit class $P/poly$. We remark that $(Samp, \mathcal{D})$ are two algorithms that pass state, which can be viewed equivalently as a single stateful algorithm $\mathcal{B}$. In our proofs we employ the latter approach, although here we state the definition as it appears in prior work.

2.3 iO-Compatible Primitives

In this section, we define extensions of some cryptographic primitives that makes them 'compatible' with indistinguishability obfuscation. [7] All of the primitives described here can be constructed from $i\mathcal{O}$ and one way functions. Their constructions can be found in [KLW15].

2.3.1 Splittable Signatures

A splittable signature scheme is a normal deterministic signature scheme, augmented by some additional algorithms and properties that we require for our application. Such a signature scheme has four different kinds of signing/verification key pairs. First, we have the standard signing/verification key pairs, where the signing key can compute signatures on any message, and the verification key can verify signatures corresponding to any message. Next, we have 'all-but-one' signing/verification keys. These keys, which correspond to a special message m^*, work for all messages except m^*; that is, the signing key can sign all messages except m^*, and the verification key can verify signatures for all messages except m^* (it does not accept any signature corresponding to m^*). Third, we have 'one' signing/verification keys. These keys correspond to a special message m', and can only be used to sign/verify signatures for m'. For all other messages, the verification algorithm does not accept any signatures. Finally, we have the rejection verification key which does not accept any signatures. The setup algorithm outputs a standard signing/verification key together with a rejection verification key, while a 'splitting' algorithm uses a standard signing key to generate 'all-but-one' and 'one' signing/verification keys.

At a high level, we require the following security properties. First, the standard verification key and the rejection verification key must be computationally indistinguishable. Intuitively, this is possible because an adversary does not have any secret key or signatures. Next, we require that if an adversary is given an 'all-but-one' secret key for message m^*, then he/she cannot distinguish between a standard verification key and an 'all-but-one' verification key corresponding to m^*. We also have a similar property for the 'one' keys. No PPT adversary, given a 'one' signing key, can distinguish between a standard verification key and a 'one' verification key. Finally, we have the 'splittability' property, which states that the keys generated by splitting one signing key are indistinguishable from the case where the 'all-but-one' key pair and the 'one' key pair are generated from different signing keys.

We will now formally describe the syntax and correctness/security properties of splittable signatures.

Syntax: A splittable signature scheme $\mathcal{S}$ for message space $\mathcal{M}$ consists of the following algorithms:

$\mathsf{Setup\text{-}Spl}(1^\lambda)$ The setup algorithm is a randomized algorithm that takes as input the security parameter λ and outputs a signing key SK, a verification key VK and *reject-verification key* $\mathrm{VK}_{\mathrm{rej}}$.

$\mathsf{Sign\text{-}Spl}(\mathrm{SK}, m)$ The signing algorithm is a deterministic algorithm that takes as input a signing key SK and a message $m \in \mathcal{M}$. It outputs a signature σ.

$\mathsf{Verify\text{-}Spl}(\mathrm{VK}, m, \sigma)$ The verification algorithm is a deterministic algorithm that takes as input a verification key VK, signature σ and a message m. It outputs either 0 or 1.

$\mathsf{Split}(\mathrm{SK}, m^*)$ The splitting algorithm is randomized. It takes as input a secret key SK and a message $m^* \in \mathcal{M}$. It outputs a signature $\sigma_{\mathrm{one}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}, m^*)$, a one-message verification key $\mathrm{VK}_{\mathrm{one}}$, an all-but-one signing key $\mathrm{SK}_{\mathrm{abo}}$ and an all-but-one verification key $\mathrm{VK}_{\mathrm{abo}}$.

$\mathsf{Sign\text{-}Spl\text{-}abo}(\mathrm{SK}_{\mathrm{abo}}, m)$ The all-but-one signing algorithm is deterministic. It takes as input an all-but-one signing key $\mathrm{SK}_{\mathrm{abo}}$ and a message m, and outputs a signature σ.

[7] In Appendix A.1, we describe a toy example to illustrate why we need to extend/modify certain primitives in order to use them with $i\mathcal{O}$.

Correctness Let $m^* \in \mathcal{M}$ be any message. Let $(\mathrm{SK}, \mathrm{VK}, \mathrm{VK}_{\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and $(\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}}) \leftarrow \mathsf{Split}(\mathrm{SK}, m^*)$. Then, we require the following correctness properties:

1. For all $m \in \mathcal{M}$, $\mathsf{Verify\text{-}Spl}(\mathrm{VK}, m, \mathsf{Sign\text{-}Spl}(\mathrm{SK}, m)) = 1$.
2. For all $m \in \mathcal{M}, m \neq m^*$, $\mathsf{Sign\text{-}Spl}(\mathrm{SK}, m) = \mathsf{Sign\text{-}Spl\text{-}abo}(\mathrm{SK}_{\mathrm{abo}}, m)$.
3. For all σ, $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{\mathrm{one}}, m^*, \sigma) = \mathsf{Verify\text{-}Spl}(\mathrm{VK}, m^*, \sigma)$.
4. For all $m \neq m^*$ and σ, $\mathsf{Verify\text{-}Spl}(\mathrm{VK}, m, \sigma) = \mathsf{Verify\text{-}Spl}(\mathrm{VK}_{\mathrm{abo}}, m, \sigma)$.
5. For all $m \neq m^*$ and σ, $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{\mathrm{one}}, m, \sigma) = 0$.
6. For all σ, $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{\mathrm{abo}}, m^*, \sigma) = 0$.
7. For all σ and all $m \in \mathcal{M}$, $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{\mathrm{rej}}, m, \sigma) = 0$.

Security We will now define the security notions for splittable signature schemes. Each security notion is defined in terms of a security game between a challenger and an adversary $\mathcal{A}$.

Definition 2.2 ($\mathrm{VK}_{\mathrm{rej}}$ indistinguishability)**.** A splittable signature scheme $\mathcal{S}$ is said to be $\mathrm{VK}_{\mathrm{rej}}$ indistinguishable if any PPT adversary $\mathcal{A}$ has negligible advantage in the following security game:

$\mathsf{Exp}\text{-}\mathrm{VK}_{\mathrm{rej}}(1^\lambda, \mathcal{S}, \mathcal{A})$:

1. Challenger computes $(\mathrm{SK}, \mathrm{VK}, \mathrm{VK}_{\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ .Next, it chooses $b \leftarrow \{0, 1\}$. If $b = 0$, it sends VK to $\mathcal{A}$. Else, it sends $\mathrm{VK}_{\mathrm{rej}}$.
2. $\mathcal{A}$ sends its guess b'.

$\mathcal{A}$ wins if $b = b'$.

We note that in the game above, $\mathcal{A}$ never receives any signatures and has no ability to produce them. This is why the difference between VK and $\mathrm{VK}_{\mathrm{rej}}$ cannot be tested.

Definition 2.3 ($\mathrm{VK}_{\mathrm{one}}$ indistinguishability)**.** A splittable signature scheme $\mathcal{S}$ is said to be $\mathrm{VK}_{\mathrm{one}}$ indistinguishable if any PPT adversary $\mathcal{A}$ has negligible advantage in the following security game:

$\mathsf{Exp}\text{-}\mathrm{VK}_{\mathrm{one}}(1^\lambda, \mathcal{S}, \mathcal{A})$:

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger computes $(\mathrm{SK}, \mathrm{VK}, \mathrm{VK}_{\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$. Next, it computes $(\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}}) \leftarrow \mathsf{Split}(\mathrm{SK}, m^*)$. It chooses $b \leftarrow \{0, 1\}$. If $b = 0$, it sends $(\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}})$ to $\mathcal{A}$. Else, it sends $(\sigma_{\mathrm{one}}, \mathrm{VK})$ to $\mathcal{A}$.
3. $\mathcal{A}$ sends its guess b'.

$\mathcal{A}$ wins if $b = b'$.

We note that in the game above, $\mathcal{A}$ only receives the signature σ_{one} on m^*, on which VK and VK_{one} behave identically.

Definition 2.4 ($\mathrm{VK}_{\mathrm{abo}}$ indistinguishability)**.** A splittable signature scheme $\mathcal{S}$ is said to be $\mathrm{VK}_{\mathrm{abo}}$ indistinguishable if any PPT adversary $\mathcal{A}$ has negligible advantage in the following security game:

$\mathsf{Exp}\text{-}\mathrm{VK}_{\mathrm{abo}}(1^\lambda, \mathcal{S}, \mathcal{A})$:

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.

2. Challenger computes $(\mathrm{SK}, \mathrm{VK}, \mathrm{VK}_{\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$. Next, it computes $(\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}}) \leftarrow \mathsf{Split}(\mathrm{SK}, m^*)$. It chooses $b \leftarrow \{0,1\}$. If $b = 0$, it sends $(\mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}})$ to $\mathcal{A}$. Else, it sends $(\mathrm{SK}_{\mathrm{abo}}, \mathrm{VK})$ to $\mathcal{A}$.
3. $\mathcal{A}$ sends its guess b'.

$\mathcal{A}$ wins if $b = b'$.

We note that in the game above, $\mathcal{A}$ does not receive or have the ability to create a signature on m^*. For all signatures $\mathcal{A}$ can create by signing with $\mathrm{SK}_{\mathrm{abo}}$, $\mathrm{VK}_{\mathrm{abo}}$ and VK will behave identically.

Definition 2.5 (Splitting indistinguishability)**.** A splittable signature scheme $\mathcal{S}$ is said to be splitting indistinguishable if any PPT adversary $\mathcal{A}$ has negligible advantage in the following security game:

$\mathsf{Exp\text{-}Spl}(1^\lambda, \mathcal{S}, \mathcal{A})$:

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger computes $(\mathrm{SK}, \mathrm{VK}, \mathrm{VK}_{\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$, $(\mathrm{SK}', \mathrm{VK}', \mathrm{VK}'_{\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$. Next, it computes $(\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}}) \leftarrow$ $\mathsf{Split}(\mathrm{SK}, m^*)$, $(\sigma'_{\mathrm{one}}, \mathrm{VK}'_{\mathrm{one}}, \mathrm{SK}'_{\mathrm{abo}}, \mathrm{VK}'_{\mathrm{abo}}) \leftarrow \mathsf{Split}(\mathrm{SK}', m^*)$. . It chooses $b \leftarrow \{0,1\}$. If $b = 0$, it sends $(\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}})$ to $\mathcal{A}$. Else, it sends $(\sigma'_{\mathrm{one}}, \mathrm{VK}'_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}})$ to $\mathcal{A}$.
3. $\mathcal{A}$ sends its guess b'.

$\mathcal{A}$ wins if $b = b'$.

In the game above, $\mathcal{A}$ is either given a system of $\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}}$ generated together by one call of $\mathsf{Setup\text{-}Spl}$ or a "split" system of $(\sigma'_{\mathrm{one}}, \mathrm{VK}'_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}})$ where the all but one keys are generated separately from the signature and key for the one message m^*. Since the correctness conditions do not link the behaviors for the all but one keys and the one message values, this split generation is not detectable by testing verification for the σ_{one} that $\mathcal{A}$ receives or for any signatures that $\mathcal{A}$ creates honestly by signing with $\mathrm{SK}_{\mathrm{abo}}$.

2.3.2 Positional Accumulators

An accumulator can be seen as a special hash function mapping unbounded[8] length strings to fixed length strings. It has two additional properties: succinct verifiability and succinct updatability. Let $\mathsf{Hash\text{-}Acc}(\cdot)$ be the hash function mapping $x = x_1 \dots x_n$ to y. Then, succinct verifiability means that there exists a 'short' proof π to prove that bit x_i is present at the i^{th} position of x. Note that this verification only requires the hash value y and the short proof π. Succinct updatability means that given y, a bit x'_i, position i and some 'short' auxiliary information, one can update y to obtain $y' = \mathsf{Hash\text{-}Acc}(x_1 \dots x'_i \dots x_n)$. We will refer to y as the tape, and x_i the symbol written at position i. For notational simplicity in our construction, we have an algorithm $\mathsf{Prep\text{-}Read}$ that takes the database and INDEX as inputs, and outputs the message at INDEX, together with a succinct proof π. Similarly, we have an algorithm $\mathsf{Prep\text{-}Write}$ that takes as input the database and INDEX, and outputs auxiliary information aux. The update algorithm can then use aux to update the accumulated hash at INDEX.

The notion of accumulators is not sufficient for using with $i\mathcal{O}$, and we need a stronger primitive called *history-less positional accumulators* that is iO-compatible. In a history-less positional accumulator, we have three different setup modes. The first one is the standard setup which outputs public parameters and the initial accumulation corresponding to the empty tape. Next, we have the read-enforced setup mode. In this mode, the algorithm takes as input the enforcing position pos, and outputs public parameters and an accumulation of the empty tape. As the name suggests, this mode is read enforcing at position pos - if the

[8] Unbounded, but polynomial in the security parameter

first k symbols written are $(\mathsf{sym}_1, \ldots, \mathsf{sym}_k)$, and their write positions are $(\mathsf{pos}_1, \ldots, \mathsf{pos}_k)$, then there *exists* exactly one opening for position pos : the correct symbol written at pos. Similarly, we have a write-enforcing setup which takes as input a position pos, and outputs public parameters and an accumulation of the empty tape. The write-enforcing property states that if $(\mathsf{sym}_i, \mathsf{pos}_i)$ are the first k writes, $\mathsf{pos} = \mathsf{pos}_k$, and acc_{k-1} is the correct accumulation after the first $k-1$ writes, then there is a unique accumulation after the k^{th} write (irrespective of the auxiliary string). Note that both the read and write enforcing properties are information theoretic. This is important when we are using these primitives with indistinguishability obfuscation. For security, we require that the different setup modes are computationally indistinguishable.

We will now give a formal description of the syntax and properties. A positional accumulator for message space $\mathcal{M}_\lambda$ consists of the following algorithms.

- $\mathsf{Setup\text{-}Acc}(1^\lambda, T) \to (\mathrm{PP}, \mathsf{acc}_0, \mathrm{STORE}_0)$ The setup algorithm takes as input a security parameter λ in unary and an integer T in binary representing the maximum number of values that can stored. It outputs public parameters PP, an initial accumulator value acc_0, and an initial storage value STORE_0.
- $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, \mathrm{INDEX}^*) \to (\mathrm{PP}, \mathsf{acc}_0, \mathrm{STORE}_0)$: The setup enforce read algorithm takes as input a security parameter λ in unary, an integer T in binary representing the maximum number of values that can be stored, and an additional INDEX^* also between 0 and $T-1$. It outputs public parameters PP, an initial accumulator value acc_0, and an initial storage value STORE_0.
- $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}(1^\lambda, T, \mathrm{INDEX}_k) \to (\mathrm{PP}, \mathsf{acc}_0, \mathrm{STORE}_0)$: The setup enforce write algorithm takes as input a security parameter λ in unary, an integer T in binary representing the maximum number of values that can be stored, and a sequence of symbol, index pairs, where each index is between 0 and $T-1$. It outputs public parameters PP, an initial accumulator value acc_0, and an initial storage value STORE_0.
- $\mathsf{Prep\text{-}Read}(\mathrm{PP}, \mathrm{STORE}_{in}, \mathrm{INDEX}) \to (m, \pi)$: The prep-read algorithm takes as input the public parameters PP, a storage value STORE_{in}, and an index between 0 and $T-1$. It outputs a symbol m (that can be ϵ) and a value π.
- $\mathsf{Prep\text{-}Write}(\mathrm{PP}, \mathrm{STORE}_{in}, \mathrm{INDEX}) \to aux$: The prep-write algorithm takes as input the public parameters PP, a storage value STORE_{in}, and an index between 0 and $T-1$. It outputs an auxiliary value aux.
- $\mathsf{Verify\text{-}Read}(\mathrm{PP}, \mathsf{acc}_{in}, m_{read}, \mathrm{INDEX}, \pi) \to \{True, False\}$: The verify-read algorithm takes as input the public parameters PP, an accumulator value acc_{in}, a symbol, m_{read}, an index between 0 and $T-1$, and a value π. It outputs $True$ or $False$.
- $\mathsf{Write\text{-}Store}(\mathrm{PP}, \mathrm{STORE}_{in}, \mathrm{INDEX}) \to \mathrm{STORE}_{out}$: The write-store algorithm takes in the public parameters, a storage value STORE_{in}, an index between 0 and $T-1$. It outputs a storage value STORE_{out}.
- $\mathsf{Update}(\mathrm{PP}, \mathsf{acc}_{in}, m_{write}, \mathrm{INDEX}, aux) \to \mathsf{acc}_{out}$ or $Reject$: The update algorithm takes in the public parameters PP, an accumulator value acc_{in}, a symbol m_{write}, and index between 0 and $T-1$, and an auxiliary value aux. It outputs an accumulator value acc_{out} or $Reject$.

In general we will think of the $\mathsf{Setup\text{-}Acc}$ algorithm as being randomized and the other algorithms as being deterministic. However, one could consider non-deterministic variants.

Correctness We consider any sequence $(m_1, \mathrm{INDEX}_1), \ldots, (m_k, \mathrm{INDEX}_k)$ of symbols $m_1, \ldots, m_k$ and indices $\mathrm{INDEX}_1, \ldots, \mathrm{INDEX}_k$ each between 0 and $T-1$. We fix any $\mathrm{PP}, \mathsf{acc}_0, \mathrm{STORE}_0 \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. For j from 1 to k, we define STORE_j iteratively as $\mathrm{STORE}_j := \mathsf{Write\text{-}Store}(\mathrm{PP}, \mathrm{STORE}_{j-1}, \mathrm{INDEX}_j, m_j)$. We similarly define aux_j and acc_j iteratively as $aux_j := \mathsf{Prep\text{-}Write}(\mathrm{PP}, \mathrm{STORE}_{j-1}, \mathrm{INDEX}_j)$ and $\mathsf{acc}_j := \mathsf{Update}(\mathrm{PP}, \mathsf{acc}_{j-1}, m_j, \mathrm{INDEX}_j, aux_j)$. Note that the algorithms other than $\mathsf{Setup\text{-}Acc}$ are deterministic, so these definitions fix precise values, not random values (conditioned on the fixed starting values $\mathrm{PP}, \mathsf{acc}_0, \mathrm{STORE}_0$).

We require the following correctness properties:

1. For every INDEX between 0 and $T-1$, Prep-Read$(\text{PP}, \text{STORE}_k, \text{INDEX})$ returns m_i, π, where i is the largest value in $[k]$ such that $\text{INDEX}_i = \text{INDEX}$. If no such value exists, then $m_i = \epsilon$.

2. For any INDEX, let $(m, \pi) \leftarrow$ Prep-Read$(\text{PP}, \text{STORE}_k, \text{INDEX})$. Then Verify-Read$(\text{PP}, \mathsf{acc}_k, m, \text{INDEX}, \pi) = True$.

Remarks on Efficiency In our construction, all algorithms will run in time polynomial in their input sizes. More precisely, Setup-Acc will be polynomial in λ and $\log(T)$. Also, accumulator and π values should have size polynomial in λ and $\log(T)$, so Verify-Read and Update will also run in time polynomial in λ and $\log(T)$. Storage values will have size polynomial in the number of values stored so far. Write-Store, Prep-Read, and Prep-Write will run in time polynomial in λ and T.

Security Let Acc = (Setup-Acc, Setup-Acc-Enforce-Read, Setup-Acc-Enforce-Write, Prep-Read, Prep-Write, Verify-Read, Write-Store, Update) be a positional accumulator for symbol set $\mathcal{M}$. We require Acc to satisfy the following notions of security.

Definition 2.6 (Indistinguishability of Read Setup)**.** A positional accumulator Acc is said to satisfy indistinguishability of read setup if any PPT adversary $\mathcal{A}$'s advantage in the security game Exp-Setup-Acc$(1^\lambda, \mathsf{Acc}, \mathcal{A})$ is at most negligible in λ, where Exp-Setup-Acc is defined as follows.

Exp-Setup-Acc$(1^\lambda, \mathsf{Acc}, \mathcal{A})$

1. Adversary chooses a bound $T \in \Theta(2^\lambda)$ and sends it to challenger.
2. $\mathcal{A}$ sends message $m \in \mathcal{M}$ and index $\text{INDEX} \in \{0, \ldots, T-1\}$ to the challenger.
3. The challenger chooses a bit b. If $b = 0$, the challenger outputs $(\text{PP}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow$ Setup-Acc$(1^\lambda, T)$. Else, it outputs $(\text{PP}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow$ Setup-Acc-Enforce-Read$(1^\lambda, T, m, \text{INDEX})$.
4. $\mathcal{A}$ sends a bit b'.

$\mathcal{A}$ wins the security game if $b = b'$.

Definition 2.7 (Indistinguishability of Write Setup)**.** A positional accumulator Acc is said to satisfy indistinguishability of write setup if any PPT adversary $\mathcal{A}$'s advantage in the security game Exp-Setup-Acc$(1^\lambda, \mathsf{Acc}, \mathcal{A})$ is at most negligible in λ, where Exp-Setup-Acc is defined as follows.

Exp-Setup-Acc$(1^\lambda, \mathsf{Acc}, \mathcal{A})$

1. Adversary chooses a bound $T \in \Theta(2^\lambda)$ and sends it to challenger.
2. $\mathcal{A}$ sends message $m \in \mathcal{M}$ and indices $\text{INDEX} \in \{0, \ldots, T-1\}$ to the challenger.
3. The challenger chooses a bit b. If $b = 0$, the challenger outputs $(\text{PP}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow$ Setup-Acc$(1^\lambda, T)$. Else, it outputs $(\text{PP}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow$ Setup-Acc-Enforce-Write$(1^\lambda, T, m, \text{INDEX})$.
4. $\mathcal{A}$ sends a bit b'.

$\mathcal{A}$ wins the security game if $b = b'$.

Definition 2.8 (Read Enforcing)**.** Consider any $\lambda \in \mathbb{N}$, $T \in \Theta(2^\lambda)$, $m_1, \ldots, m_k \in \mathcal{M}$, $\text{INDEX}_1, \ldots, \text{INDEX}_k \in \{0, \ldots, T-1\}$ and any $\text{INDEX}^* \in \{0, \ldots, T-1\}$.

Let $(\text{PP}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow$ Setup-Acc-Enforce-Read$(1^\lambda, T, m, \text{INDEX}), \text{INDEX}^*)$. For j from 1 to k, we define STORE_j iteratively as $\text{STORE}_j :=$ Write-Store$(\text{PP}, \text{STORE}_{j-1}, \text{INDEX}_j, m_j)$. We similarly define aux_j and acc_j iteratively as $aux_j :=$ Prep-Write$(\text{PP}, \text{STORE}_{j-1}, \text{INDEX}_j)$ and $\mathsf{acc}_j :=$ Update$(\text{PP}, \mathsf{acc}_{j-1}, m_j, \text{INDEX}_j, aux_j)$. Acc is said to be *read enforcing* if Verify-Read$(\text{PP}, \mathsf{acc}_k, m, \text{INDEX}^*, \pi) = True$, then either $\text{INDEX}^* \notin \{\text{INDEX}_1, \ldots, \text{INDEX}_k\}$ and $m = \epsilon$, or $m = m_i$ for the largest $i \in [k]$ such that $\text{INDEX}_i = \text{INDEX}^*$. Note that this is an information-theoretic property: we are requiring that for all other symobls m, values of π that would cause Verify-Read to output $True$ at INDEX^* do no exist.

Definition 2.9 (Write Enforcing)**.** Consider any $\lambda \in \mathbb{N}$, $T \in \Theta(2^\lambda)$, $m_1, \ldots, m_k \in \mathcal{M}$, $\text{INDEX}_1, \ldots, \text{INDEX}_k \in \{0, \ldots, T-1\}$. Let $(\text{PP}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}(1^\lambda, T, \text{INDEX})$. For j from 1 to k, we define STORE_j iteratively as $\text{STORE}_j := \mathsf{Write\text{-}Store}(\text{PP}, \text{STORE}_{j-1}, \text{INDEX}_j, m_j)$. We similarly define aux_j and acc_j iteratively as $aux_j := \mathsf{Prep\text{-}Write}(\text{PP}, \text{STORE}_{j-1}, \text{INDEX}_j)$ and $\mathsf{acc}_j := \mathsf{Update}(\text{PP}, \mathsf{acc}_{j-1}, m_j, \text{INDEX}_j, aux_j)$. Acc is said to be *write enforcing* if $\mathsf{Update}(\text{PP}, \mathsf{acc}_{k-1}, m_k, \text{INDEX}_k, aux) = \mathsf{acc}_{out} \neq \mathit{Reject}$, for any aux, then $\mathsf{acc}_{out} = \mathsf{acc}_k$. Note that this is an information-theoretic property: we are requiring that an aux value producing an accumulated value other than acc_k or *Reject* does not exist.

2.3.3 Iterators

In this section, we define the notion of *cryptographic iterators*. A cryptographic iterator essentially consists of a small state that is updated in an iterative fashion as messages are received. An update to apply a new message given current state is performed via some public parameters.

Since states will remain relatively small regardless of the number of messages that have been iteratively applied, there will in general be many sequences of messages that can lead to the same state. However, our security requirement will capture that the normal public parameters are computationally indistinguishable from specially constructed "enforcing" parameters that ensure that a particular *single* state can be only be obtained as an output as an update to precisely one other state, message pair. Note that this enforcement is a very localized property to a particular state, and hence can be achieved information-theoretically when we fix ahead of time where exactly we want this enforcement to be.

Syntax Let ℓ be any polynomial. An iterator $\mathcal{I}$ with message space $\mathcal{M}_\lambda = \{0,1\}^{\ell(\lambda)}$ and state space $\mathcal{S}_\lambda$ consists of three algorithms - $\mathsf{Setup\text{-}Itr}$, $\mathsf{Setup\text{-}Itr\text{-}Enforce}$ and $\mathsf{Iterate}$ defined below.

$\mathsf{Setup\text{-}Itr}(1^\lambda, T)$ The setup algorithm takes as input the security parameter λ (in unary), and an integer bound T (in binary) on the number of iterations. It outputs public parameters PP and an initial state $v_0 \in \mathcal{S}_\lambda$.

$\mathsf{Setup\text{-}Itr\text{-}Enforce}(1^\lambda, T, \mathbf{m} = (m_1, \ldots, m_k))$ The enforced setup algorithm takes as input the security parameter λ (in unary), an integer bound T (in binary) and k messages $(m_1, \ldots, m_k)$, where each $m_i \in \{0,1\}^{\ell(\lambda)}$ and k is some polynomial in λ. It outputs public parameters PP and a state $v_0 \in \mathcal{S}$.

$\mathsf{Iterate}(\text{PP}, v_{\mathsf{in}}, m)$ The iterate algorithm takes as input the public parameters PP, a state v_{in}, and a message $m \in \{0,1\}^{\ell(\lambda)}$. It outputs a state $v_{\mathsf{out}} \in \mathcal{S}_\lambda$.

For simplicity of notation, we will drop the dependence of ℓ on λ. Also, for any integer $k \leq T$, we will use the notation $\mathsf{Iterate}^k(\text{PP}, v_0, (m_1, \ldots, m_k))$ to denote $\mathsf{Iterate}(\text{PP}, v_{k-1}, m_k)$, where $v_j = \mathsf{Iterate}(\text{PP}, v_{j-1}, m_j)$ for all $1 \leq j \leq k-1$.

Security Let $\mathcal{I} = (\mathsf{Setup\text{-}Itr}, \mathsf{Setup\text{-}Itr\text{-}Enforce}, \mathsf{Iterate})$ be an interator with message space $\{0,1\}^\ell$ and state space $\mathcal{S}_\lambda$. We require the following notions of security.

Definition 2.10 (Indistinguishability of Setup)**.** An iterator $\mathcal{I}$ is said to satisfy indistinguishability of Setup phase if any PPT adversary $\mathcal{A}$'s advantage in the security game $\mathsf{Exp\text{-}Setup\text{-}Itr}(1^\lambda, \mathcal{I}, \mathcal{A})$ at most is negligible in λ, where $\mathsf{Exp\text{-}Setup\text{-}Itr}$ is defined as follows.

$\mathsf{Exp\text{-}Setup\text{-}Itr}(1^\lambda, \mathcal{I}, \mathcal{A})$

1. The adversary $\mathcal{A}$ chooses a bound $T \in \Theta(2^\lambda)$ and sends it to challenger.
2. $\mathcal{A}$ sends k messages $m_1, \ldots, m_k \in \{0,1\}^\ell$ to the challenger.
3. The challenger chooses a bit b. If $b = 0$, the challenger outputs $(\text{PP}, v_0) \leftarrow \mathsf{Setup\text{-}Itr}(1^\lambda, T)$. Else, it outputs $(\text{PP}, v_0) \leftarrow \mathsf{Setup\text{-}Itr\text{-}Enforce}(1^\lambda, T, 1^k, \mathbf{m} = (m_1, \ldots, m_k))$.
4. $\mathcal{A}$ sends a bit b'.

$\mathcal{A}$ wins the security game if $b = b'$.

Definition 2.11 (Enforcing)**.** Consider any $\lambda \in \mathbb{N}$, $T \in \Theta(2^\lambda)$, $k < T$ and $m_1, \ldots, m_k \in \{0,1\}^\ell$. Let $(\mathrm{PP}, v_0) \leftarrow \mathsf{Setup\text{-}Itr\text{-}Enforce}(1^\lambda, T, \mathbf{m} = (m_1, \ldots, m_k))$ and $v_j = \mathsf{Iterate}^j(\mathrm{PP}, v_0, (m_1, \ldots, m_j))$ for all $1 \leq j \leq k$. Then, $\mathcal{I} = (\mathsf{Setup\text{-}Itr}, \mathsf{Setup\text{-}Itr\text{-}Enforce}, \mathsf{Iterate})$ is said to be *enforcing* if

$$v_k = \mathsf{Iterate}(\mathrm{PP}, v', m') \implies (v', m') = (v_{k-1}, m_k).$$

Note that this is an information-theoretic property.

2.4 Attribute Based Encryption

An ABE scheme where policies are represented by Turing machines comprises of the following four algorithms $(\mathsf{ABE.setup}, \mathsf{ABE.enc}, \mathsf{ABE.keygen}, \mathsf{ABE.dec})$:

- $\mathsf{ABE.setup}(1^\lambda) \to (\mathrm{PK}_{\mathsf{ABE}}, \mathrm{MSK}_{\mathsf{ABE}})$: The setup algorithm takes as input the security parameter λ and outputs the public key $\mathrm{PK}_{\mathsf{ABE}}$ and the master secret key $\mathrm{MSK}_{\mathsf{ABE}}$
- $\mathsf{ABE.enc}(m, x, \mathrm{PK}_{\mathsf{ABE}}) \to \mathsf{ct}$: The encryption algorithm takes as input the message m, the attribute string x of unbounded length and the public key $\mathrm{PK}_{\mathsf{ABE}}$ and it outputs the corresponding ciphertext ct_x specific to the attribute string.
- $\mathsf{ABE.keygen}(\mathrm{MSK}_{\mathsf{ABE}}, M) \to \mathrm{SK}\{M\}$: The key generation algorithm takes as input $\mathrm{MSK}_{\mathsf{ABE}}$ and a Turing machine M and outputs the secret key $\mathrm{SK}\{M\}$ specific to M
- $\mathsf{ABE.dec}(\mathrm{SK}\{M\}, \mathsf{ct}) \to m$ or $\perp$: The decryption algorithm takes in $\mathrm{SK}\{M\}$ and ciphertext ct and outputs either a message m or $\perp$.

The *correctness* of the scheme guarantees that if $\mathsf{ABE.enc}(m, x, \mathrm{PK}_{\mathsf{ABE}}) \to \mathsf{ct}_x$ and $\mathsf{ABE.keygen}(\mathrm{MSK}_{\mathsf{ABE}}, M) \to \mathrm{SK}\{M\}$ then $\mathsf{ABE.dec}(\mathrm{SK}\{M\}, \mathsf{ct}_x) \to m$.

2.5 Selective Security

Consider the following experiment between a challenger $\mathcal{C}$ and a stateful adversary $\mathcal{A}$:

- **Setup Phase**: $\mathcal{A}$ sends the challenge attribute string x^* of his choice to $\mathcal{C}$. $\mathcal{C}$ runs the $\mathsf{ABE.setup}(1^\lambda)$ and sends across $\mathrm{PK}_{\mathsf{ABE}}$ to $\mathcal{A}$.
- **Pre-Challenge Query Phase**: $\mathcal{A}$ gets to query for secret keys corresponding to Turing machines. For each query M such that $M(x^*) = 0$, the challenger computes $\mathrm{SK}\{M\} \leftarrow \mathsf{ABE.keygen}(\mathrm{MSK}_{\mathsf{ABE}}, .)$ and sends it to $\mathcal{A}$.
- **Challenge Phase** $\mathcal{A}$ sends two messages m_0, m_1 with $|m_0| = |m_1|$, the challenger chooses bit b uniformly at random and outputs $\mathsf{ct}^* = \mathsf{ABE.enc}(m_b, x^*, \mathrm{PK}_{\mathsf{ABE}})$.
- **Post-Challenge Query Phase**: This is identical to the Pre-Challenge Phase.
- **Guess**: Finally, $\mathcal{A}$ sends its guess b' and wins if $b = b'$.

The advantage of $\mathcal{A}$, $\mathsf{Adv}^{ABE}_{\mathcal{A}}(\lambda)$ in the above experiment is defined to be $|\Pr[b' = b] - \frac{1}{2}|$.

Definition 2.12. An ABE scheme is said to be *selectively secure* if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{ABE}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.

3 Constrained Pseudorandom Functions for Turing Machines

The notion of constrained pseudorandom functions was introduced in the concurrent works of [BW13, BGI14, KPTZ13]. Informally, a constrained PRF extends the notion of standard PRFs, enabling the master PRF key holder to compute 'constrained keys' that allow PRF evaluations on certain inputs, while the PRF evaluation on remaining inputs 'looks' random. In the above mentioned works, these constraints could only handle bounded length inputs. In order to allow unbounded inputs, we need to ensure that the constrained keys correspond to polynomial time Turing Machines. A formal definition is as follows.

Let $\mathcal{M}_\lambda$ be a family of Turing machines with (worst case) running time bounded by 2^λ. Let $\mathcal{K}$ denote the key space, $\mathcal{X}$ the input domain and $\mathcal{Y}$ the range space. A pseudorandom $\mathsf{PRF} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is said to be *constrained* with respect to the Turing machine family $\mathcal{M}_\lambda$ if there is an additional key space $\mathcal{K}_c$, and three algorithms $\mathsf{PRF.setup}$, $\mathsf{PRF.constrain}$ and $\mathsf{PRF.eval}$ as follows:

- $\mathsf{PRF.setup}(1^\lambda)$ is a PPT algorithm that takes the security parameter λ as input and outputs a key $K \in \mathcal{K}$.
- $\mathsf{PRF.constrain}(K, M)$ is a PPT algorithm that takes as input a PRF key $K \in \mathcal{K}$ and a Turing machine $M \in \mathcal{M}_\lambda$ and outputs a constrained key $K\{M\} \in \mathcal{K}_c$.
- $\mathsf{PRF.eval}(K\{M\}, x)$ is a deterministic polynomial time algorithm that takes as input a constrained key $K\{M\} \in \mathcal{K}_c$ and $x \in \mathcal{X}$ and outputs an element $y \in \mathcal{Y}$. Let $K\{M\}$ be the output of $\mathsf{PRF.constrain}(K, M)$. For correctness, we require the following:

$$\mathsf{PRF.eval}(K\{M\}, x) = F(K, x) \text{ if } M(x) = 1.$$

For simplicity of notation, we will use $\mathsf{PRF}(K\{M\}, x)$ to denote $\mathsf{PRF.eval}(K\{M\}, x)$.

3.1 Security of Constrained Pseudorandom Functions

Intuitively, we require that even after obtaining several constrained keys, no polynomial time adversary can distinguish a truly random string from the PRF evaluation at a point not accepted by the queried Turing machines. In this work, we achieve a weaker notion of security called *selective security*, which is formalized by the following security game between a challenger and an adversary Att.

Let $\mathsf{PRF} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a constrained PRF with respect to a Turing machine family $\mathcal{M}$. The security game consists of three phases.

Setup Phase The adversary sends the challenge input x^*. The challenger chooses a random key $K \leftarrow \mathcal{K}$ and a random bit $b \leftarrow \{0, 1\}$. If $b = 0$, the challenger outputs $\mathsf{PRF}(K, x^*)$. Else, the challenger outputs a random element $y \leftarrow \mathcal{Y}$.

Query Phase In this phase, Att is allowed to ask for the following queries:

- **Evaluation Query** Att sends $x \in \mathcal{X}$, and receives $\mathsf{PRF}(K, x)$.
- **Key Query** Att sends a Turing machine $M \in \mathcal{M}$ such that $M(x^*) = 0$, and receives $\mathsf{PRF.constrain}(K, M)$.

Guess Finally, A outputs a guess b' of b.

A wins if $b = b'$ and the advantage of Att is defined to be $\mathsf{Adv}_{\mathsf{Att}}(\lambda) = \left| \Pr[\mathsf{Att} \text{ wins}] - 1/2 \right|$.

Definition 3.1. The pseudorandom function PRF is a secure constrained PRF with respect to $\mathcal{M}$ if for all PPT adversaries A $\mathsf{Adv}_{\mathsf{Att}}(\lambda)$ is negligible in λ.

3.2 Puncturable Pseudorandom Functions

A special class of constrained PRFs, called *puncturable PRFs*, was introduced in the work of [SW14]. In a puncturable PRF, the constrained key queries correspond to points in the input domain, and the constrained key is one that allows PRF evaluations at all points except the punctured point.

Formally, a PRF $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a puncturable pseudorandom function if there is an additional key space $\mathcal{K}_p$ and three polynomial time algorithms $\mathsf{F.setup}$, $\mathsf{F.eval}$ and $\mathsf{F.puncture}$ as follows:

- $\mathsf{F.setup}(1^\lambda)$ is a randomized algorithm that takes the security parameter λ as input and outputs a description of the key space $\mathcal{K}$, the punctured key space $\mathcal{K}_p$ and the PRF F.
- $\mathsf{F.puncture}(K, x)$ is a randomized algorithm that takes as input a PRF key $K \in \mathcal{K}$ and $x \in \mathcal{X}$, and outputs a key $K_x \in \mathcal{K}_p$.
- $\mathsf{F.eval}(K_x, x')$ is a deterministic algorithm that takes as input a punctured key $K_x \in \mathcal{K}_p$ and $x' \in \mathcal{X}$. Let $K \in \mathcal{K}$, $x \in \mathcal{X}$ and $K_x \leftarrow \mathsf{F.puncture}(K, x)$. For correctness, we need the following property:

$$\mathsf{F.eval}(K_x, x') = \begin{cases} \mathsf{F}(K, x') & \text{if } x \neq x' \\ \perp & \text{otherwise} \end{cases}$$

The selective security notion is analogous to the security notion of constrained PRFs.

4 Construction

A high level description of our construction: Our constrained PRF construction uses a puncturable PRF F as the base pseudorandom function. The setup algorithm chooses a puncturable PRF key K together with the public parameters of the accumulator and an accumulation of the empty tape (it also outputs additional parameters for the authenticity checks described in the next paragraph). To evaluate the constrained PRF on input x, one first accumulates the input x. Let y denote this accumulation. The PRF evaluation is $F(K, y)$.

Next, let us consider the constrained key for Turing machine M. The major component of this key is an obfuscated program Prog. At a very high level, this program evaluates the next-step circuit of M. Its main inputs are the time step t, hash y of the input and the symbol, state, position of TM used at step t. Using the state and symbol, it computes the next state and the symbol to be written. If the state is accepting, it outputs $F(K, y)$, else it outputs the next state and symbol. However, this is clearly not enough, since the adversary could pass illegal states and symbols as inputs. So the program first performs some additional authenticity checks, then evaluates the next (state, symbol), and finally outputs authentication required for the next step evaluation. These authenticity checks are imposed via the accumulator, signature scheme and iterator. For these checks, Prog takes additional inputs: accumulation of the current tape acc, proof π that the input symbol is the correct symbol at the tape-head position, auxiliary string aux to update the accumulation, iterated value and signature σ. The iterated value and the signature together ensure that the correct state and accumulated value is input at each step, while the accumulation ensures that the adversary cannot send a wrong symbol. Finally, to perform the 'tail-cutting', the program requires an additional input seed. The first and last step of the program are for checking the validity of seed, and to output the new seed if required. The constrained key also has another program $\mathsf{Init\text{-}Sign}$ which is used to sign the accumulation of the input. In the end, if all the checks go through, the final output will be the PRF evaluation using the constrained key.

Formal description: We now describe our constrained pseudorandom function where the constrained keys correspond to Turing machines. Let Acc = ($\mathsf{Setup\text{-}Acc}$, $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}$, $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}$, $\mathsf{Prep\text{-}Read}$, $\mathsf{Prep\text{-}Write}$, $\mathsf{Verify\text{-}Read}$, $\mathsf{Write\text{-}Store}$, Update) be a history-less positional accumulator, Itr = ($\mathsf{Setup\text{-}Itr}$, $\mathsf{Setup\text{-}Itr\text{-}Enforce}$, $\mathsf{Iterate}$) an iterator, $\mathcal{S}$ = ($\mathsf{Setup\text{-}Spl}$, $\mathsf{Sign\text{-}Spl}$, $\mathsf{Verify\text{-}Spl}$, Split, $\mathsf{Sign\text{-}Spl\text{-}abo}$)

a splittable signature scheme and $\mathrm{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ a length doubling injective pseudorandom generator.

Let F be a puncturable pseudorandom function whose domain and range are chosen appropriately, depending on the accumulator, iterator and splittable signature scheme. For simplicity, we assume that F takes inputs of bounded length, instead of fixed length inputs. This assumption can be easily removed by using different PRFs for different input lengths (in our case, we will require three different fixed-input-length PRFs). Also, to avoid confusion, the puncturable PRF keys (both master and punctured) are represented using lower case letters (e.g. k, $k\{z\}$), while the constrained PRF keys are represented using upper case letters (e.g. K, $K\{M\}$).

- $\mathsf{PRF.setup}(1^\lambda)$: The setup algorithm takes the security parameter λ as input. It first chooses a puncturable PRF keys $k \leftarrow \mathsf{F.setup}(1^\lambda)$. Next, it runs the accumulator setup to obtain $(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda)$. The master PRF key is $K = (k, \mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0)$.
- PRF Evaluation: To evaluate the PRF with key $K = (k, \mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0)$ on input $x = x_1 \ldots x_n$, first 'hash' the input using the accumulator. More formally, let $\mathsf{Hash\text{-}Acc}(x) = \mathsf{acc}_n$, where for all $j \leq n$, acc_j is defined as follows:
  - $\mathrm{STORE}_j = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{j-1}, j-1, x_j)$
  - $aux_j = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{j-1}, j-1)$
  - $\mathsf{acc}_j = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_{j-1}, x_j, j-1, aux_j)$

  The PRF evaluation is defined to be $F(k, \mathsf{Hash\text{-}Acc}(x))$.
- $\mathsf{PRF.constrain}(K = (k, \mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0), M)$: The constrain algorithm first chooses puncturable PRF keys $k_1, \ldots, k_\lambda$ and $k_{\mathrm{sig},A}$ and runs the iterator setup to obtain $(\mathrm{PP}_{\mathsf{Itr}}, \mathsf{it}_0) \leftarrow \mathsf{Setup\text{-}Itr}(1^\lambda, T)$. Next, it computes an obfuscation of program Prog (defined in Figure 1) and $\mathsf{Init\text{-}Sign}$ (defined in Figure 2). The constrained key $K\{M\} = (\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0, \mathrm{PP}_{\mathsf{Itr}}, \mathsf{it}_0, i\mathcal{O}(\mathsf{Prog}), i\mathcal{O}(\mathsf{Init\text{-}Sign}))$.
- PRF Evaluation using Constrained Key: Let $K\{M\} = (\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0, \mathrm{PP}_{\mathsf{Itr}}, \mathsf{it}_0, P_1, P_2)$ be a constrained key corresponding to machine M, and $x = x_1, \ldots, x_n$ the input. As in the evaluation using master PRF key, first compute $\mathsf{acc\text{-}inp} = \mathsf{Hash\text{-}Acc}(x)$.

  To begin the evaluation, compute a signature on the initial values using the program P_2. Let $\sigma_0 = P_2(\mathsf{acc\text{-}inp})$.

  Suppose M runs for t^* steps on input x. Run the program P_1 iteratively for t^* steps. Set $\mathsf{pos}_0 = 0$, $\mathsf{seed}_0 =$ ' ', and for $i = 1$ to t^*, compute

  1. Let $(\mathsf{sym}_{i-1}, \pi_{i-1}) = \mathsf{Prep\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{i-1}, \mathsf{pos}_{i-1})$.
  2. Compute $aux_{i-1} \leftarrow \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{i-1}, \mathsf{pos}_{i-1})$.
  3. Let $\mathrm{out} = P_1(i, \mathsf{seed}_{i-1}, \mathsf{pos}_{i-1}, \mathsf{sym}_{i-1}, \mathsf{st}_{i-1}, \mathsf{acc}_{i-1}, \pi_{i-1}, aux_{i-1}, \mathsf{acc\text{-}inp}, \mathsf{it}_{i-1}, \sigma_{i-1})$. If $j = t^*$, output out. Else, parse out as $(\mathsf{sym}_{w,i}, \mathsf{pos}_i, \mathsf{st}_i, \mathsf{acc}_i, \mathsf{it}_i, \sigma_i, \mathsf{seed}_i)$.
  4. Compute $\mathrm{STORE}_i = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{i-1}, \mathsf{pos}_{i-1}, \mathsf{sym}_{w,i})$.

  The output at step t^* is the PRF evaluation using the constrained key.

Program Prog

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux, accumulation of input $\mathsf{acc\text{-}inp}$
Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) Let $r_A = \mathsf{F}(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$.
   (b) Let $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_A = \mathsf{F}(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 1: Program Prog

Program Init-Sign

Constants: Puncturable PRF key $k_{\text{sig},A}$, Initial TM state q_0, Iterator value it_0

Input: Accumulation of input $\mathsf{acc\text{-}inp}$

1. Let $F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, 0)) = r_{\text{sig}}$. Compute $(\text{SK}, \text{VK}, \text{VK}_{\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig}})$.
2. Output $\sigma = \mathsf{Sign\text{-}Spl}(\text{SK}, (\mathsf{it}_0, q_0, \mathsf{acc\text{-}inp}, 0))$.

Figure 2: Program Init-Sign

4.1 Proof of Selective Security

Proof Outline To prove security, we will first describe a sequence of hybrid experiments, and then show that consecutive hybrid experiments are computationally indistinguishable. Recall that our goal is to achieve selective security, and therefore the challenger knows the input and the accumulated hash of input $\mathsf{acc\text{-}inp}$ before receiving any constrained key queries. At a very high level, our approach is similar to the one in [BZ14] - we will replace the PRF key in the obfuscated program with one that is punctured at $\mathsf{acc\text{-}inp}$. Once this

is done, the adversary cannot distinguish between a PRF evaluation at acc-inp, and a truly random string. But before we can replace the PRF key with a punctured key, we must ensure that all these obfuscated programs output $\perp$ on inputs corresponding to acc-inp. This is the critical part here, since there exist other 'legal' inputs that may hash to acc-inp.

Our proof will consist of a sequence of computationally indistinguishable hybrid experiments. The first hybrid experiment (Hybrid_0) will correspond to the real experiment. In the next hybrid (Hybrid_1), all constrained key queries are such that they output $\perp$ on all inputs that hash to acc-inp. To show that Hybrid_0 and Hybrid_1 are computationally indistinguishable, we will use a sequence of intermediate hybrids $\mathsf{Hybrid}_{0,j}$ (the number of intermediate hybrids depends on the number of constrained key queries).

In the final hybrid (Hybrid_2), the program now has a PRF key that is punctured at acc-inp. At this point, the PPT adversary cannot distinguish between the PRF evaluation at acc-inp and a truly random string. Finally, we will argue that any PPT adversary has atmost negligible advantage in Hybrid_2.

Sequence of Hybrid Experiments We will first set up some notation for the hybrid experiments. Let q denote the number of constrained key queries made by the adversary. Let x^* denote the challenge input chosen by the adversary, $(k, \mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0)$ the master key chosen by challenger, $\mathsf{acc\text{-}inp}^* = \mathsf{Hash\text{-}Acc}(x^*)$ as defined in the construction. Let M_j denote the j^{th} constrained key query, and t_j^* be the running time of machine M_j on input x^*, and τ_j be the smallest power of two greater than t_j^*. The program Prog_j denotes the program Prog with machine M_j hardwired.

Hybrid_0 This corresponds to the real experiment.

Next, we define q hybrid experiments $\mathsf{Hybrid}_{0,j}$ for $1 \leq j \leq q$.

$\mathsf{Hybrid}_{0,j}$: Let Prog-1 denote the program defined in Figure 3. In this experiment, the challenger sends an obfuscation of the program $\mathsf{Prog\text{-}1}_i$ (Prog-1 with machine M_i hardwired) for the i^{th} query if $i \leq j$. For the remaining queries, the challenger outputs an obfuscation of Prog_i.

Hybrid_1 : This experiment is identical to hybrid $\mathsf{Hybrid}_{0,q}$. In this experiment, the challenger sends an obfuscation of $\mathsf{Prog\text{-}1}_i$ for all constrained key queries.

Hybrid_2 : In this experiment, the challenger punctures the PRF key k at input $\mathsf{acc\text{-}inp}^*$ and uses the punctured key for all key queries. More formally, after receiving the challenge input x^*, it chooses $(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda)$ and computes $\mathsf{acc\text{-}inp}^* = \mathsf{Hash\text{-}Acc}(x^*)$. It then chooses a PRF key k and computes $k\{\mathsf{acc\text{-}inp}^*\} \leftarrow \mathsf{F.puncture}(k, \mathsf{acc\text{-}inp}^*)$. Next, it receives constrained key queries for machines $M_1, \ldots, M_q$. For each query, it chooses $(\mathrm{PP}_{\mathsf{ltr}}, \mathsf{it}_0) \leftarrow \mathsf{Setup\text{-}ltr}(1^\lambda)$ and PRF keys $k_1, \ldots, k_\lambda, k_{\mathrm{sig},A}$. It computes an obfuscation of $\mathsf{Prog\text{-}1}\{M_i\ , \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{ltr}}, k\{\mathsf{acc\text{-}inp}^*\}, k_{\mathrm{sig},A}\}$.

$$\mathsf{Hybrid}_0 \equiv \mathsf{Hybrid}_{0,0} \xrightarrow{4.1} \mathsf{Hybrid}_{0,1} \xrightarrow{4.1} \cdots \xrightarrow{4.1} \mathsf{Hybrid}_{0,q} \equiv \mathsf{Hybrid}_1 \downarrow i\mathcal{O}$$

$$0 \overset{\mathrm{PPRF}}{\approx} \mathsf{Hybrid}_2$$

4.1.1 Analysis

Let $\mathsf{Adv}_i^{\mathcal{A}}$ denote the advantage of any PPT adversary $\mathcal{A}$ in the hybrid experiment Hybrid_i (similarly, let $\mathsf{Adv}_{0,j}^{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in the intermediate hybrid experiment $\mathsf{Hybrid}_{0,j}$).

Program Prog-1

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time $t^* \in [T]$
Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A} \in \mathcal{K}$
Hardwired accumulated value $\mathsf{acc\text{-}inp}^*$

Inputs: Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux, accumulation of input $\mathsf{acc\text{-}inp}$
Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) Let $r_{\text{sig}} = F(k_{\text{sig},A}, t-1)$. Compute $(\text{SK}, \text{VK}, \text{VK}_{\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig}})$.
   (b) Let $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}, \mathsf{acc\text{-}inp})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   (c) If $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
   Else If $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ output $\perp$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_{\text{sig}} = F(k_{\text{sig},A}, t)$. Compute $(\text{SK}', \text{VK}', \text{VK}'_{\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig}})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}}, \mathsf{acc\text{-}inp})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}', m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 3: Program Prog-1

Recall $\mathsf{Hybrid}_{0,0}$ corresponds to the experiment Hybrid_0, and $\mathsf{Hybrid}_{0,q}$ corresponds to the experiment Hybrid_1. Using the following lemma, we can show that $|\mathsf{Adv}_0^{\mathcal{A}} - \mathsf{Adv}_1^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Lemma 4.1. Assuming F is a puncturable PRF, Acc is a secure positional accumulator, Itr is a secure history-less positional iterator, $\mathcal{S}$ is a secure splittable signature scheme and $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_{0,j}^{\mathcal{A}} - \mathsf{Adv}_{0,j+1}^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

The proof of this lemma involves multiple hybrids, and is similar to the [KLW15] security proof for message hiding encodings. We give a high level description of the proof here, and defer the full details to Appendix B. Let M be the j^{th} query, and let us assume M outputs 0 on input x^* after t^* steps. The only difference between Prog and Prog-1 corresponding to M is that the latter, on input $\mathsf{acc\text{-}inp}^*$, never outputs $F(K, \mathsf{acc\text{-}inp}^*)$, even if the program reaches q_{ac}. Of course, if the program is executed honestly, then it does not reach the accepting state. However, we need to argue that even if the adversary feeds illegal inputs, the program does not reach q_{ac}. A natural approach would be the following: define t^* intermediate hybrid

programs, where the i^{th} hybrid program does not reach q_{ac} for the first i steps on inputs corresponding to $\mathsf{acc\text{-}inp}^*$. However, it is not clear how to implement such a 'direct' approach. Instead, we need to employ some preprocessing and post-processing hybrids.

Preprocessing hybrid: The first step is to modify the program Prog to allow additional valid signatures without being detected. In particular, we have an additional PRF key in the program, and this generates 'bad' signing/verification keys. The program first checks if the input signature is accepted by the usual 'good' verification key. If not, it checks if it is accepted by the 'bad' verification key. If the incoming signature is bad, then the output signature is also computed using the bad signing key. Let us call this hybrid Hyb-1. This switch is indistinguishable because the $\mathsf{Init\text{-}Sign}$ program only outputs a good signature, and we use the rejection-verification key indistinguishability property to show that this change is indistinguishable.

Next, we define t^* intermediate hybrid programs Hyb-$(1, i)$. These ensure that the program does not reach the accepting state for first i steps on inputs corresponding to $\mathsf{acc\text{-}inp}^*$.

Intermediate hybrids Hyb-$(1, i)$*:* In the i^{th} intermediate hybrid, the program does not output PRF evaluation if $t \leq i$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. Moreover, if $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, it only accepts good signatures for the first $i-1$ steps. For the i^{th} step, if $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, it accepts only good signatures, but outputs a bad signature if the input iterated value, accumulated value or state are not the correct ones for time step i (here, the program has the correct values for step i hardwired). We now need to go from step Hyb-$(1, i)$ to step Hyb-$(1, i+1)$.

For this, we will first ensure that if $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, the only signature accepted at step $i+1$ is the one corresponding to the correct (iterated value, accumulated value, state) input tuple at step $i+1$. Intuitively, this is true because the program, at step i, outputs a bad signature for all other tuples. To enforce this, we use the properties of the splittable signature schemes. Next, we make the accumulator read-enforcing. This would mean that both the state and symbol input at step $i+1$ are the correct ones. As a result, the program cannot output the PRF evaluation at step $i+1$ if $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. So now, the state and symbol output at step $i+1$ also have to be the correct ones. To ensure that the accumulated value and iterated value output are also correct, we make the accumulator write-enforcing and iterator enforcing respectively. Together, these will ensure that the transition from Hyb-$(1, i)$ and Hyb-$(1, i+1)$ are computationally indistinguishable.

Post-processing hybrids: Continuing this way, we can ensure, step by step, that the program does not output the PRF evaluation on $\mathsf{acc\text{-}inp}^*$. However, the approach described above will require exponential hybrids. To make the number of intermediate hybrids polynomial, we use the 'tail-cutting' technique described in Section 1. Note that the program, after t^* steps, only outputs $\perp$. Suppose t^* is a power of two. Using a PRG trick, we can wipe out steps t^* to $2t^*$ in one shot. At every step where t is a power of two, the program outputs a new PRG seed, and this PRG seed's validity is checked till t reaches the next power of two. Now, if no PRG seed is output at step t^*, then using the PRG security, one can ensure that the PRG seed validity check fails. As a result, for all $t \in (t^*, 2t^*)$, the program outputs $\perp$.

Lemma 4.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. Let us assume for now that the adversary makes exactly one constrained key query corresponding to machine M_1. This can be naturally extended to the general case via a hybrid argument.

Note that the only difference between the two hybrids is the PRF key hardwired in Prog-1. In one case, the challenger sends an obfuscation of $P_1 = \mathsf{Prog}\text{-}1\{M_1, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k, k_1, \ldots, k_\lambda, k_{\text{sig},A}\}$, while in the other, it sends an obfuscation of $P_2 = \mathsf{Prog}\text{-}1\{M_1, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k\{\mathsf{acc\text{-}inp}^*\}, k_1, \ldots, k_\lambda, k_{\text{sig},A}\}$. To prove that these two hybrids are computationally indistinguishable, it suffices to show that the P_1 and P_2 are functionally identical. Note that program P_1 computes $\mathsf{F}(k, \mathsf{acc\text{-}inp})$ only if $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$. As a result, using the correctness property of puncturable PRFs, the programs have identical functionality. ∎

Lemma 4.3. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_2^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $|\mathsf{Adv}_2^{\mathcal{A}}| = \epsilon$. We will use $\mathcal{A}$ to construct a PPT algorithm $\mathcal{B}$ that breaks the security of the puncturable PRF F.

To begin with, $\mathcal{B}$ receives the challenge input x^* from $\mathcal{A}$. It chooses $(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda)$. It then computes $\mathsf{acc\text{-}inp}^* = \mathsf{Hash\text{-}Acc}(x^*)$, and sends $\mathsf{acc\text{-}inp}^*$ to the PRF challenger as the challenge input. It receives a punctured key k' and an element y (which is either the pseudorandom evaluation at $\mathsf{acc\text{-}inp}^*$ or a truly random string in the range space). $\mathcal{B}$ sends y to $\mathcal{A}$ as the challenge response.

Next, it receives multiple constrained key requests. For the i^{th} query corresponding to machine M_i, $\mathcal{B}$ chooses PRF keys $k_1, \ldots, k_\lambda, k_{\mathrm{sig},A} \leftarrow \mathsf{F.setup}(1^\lambda)$, $(\mathrm{PP}_{\mathsf{ltr}}, \mathsf{it}_0) \leftarrow \mathsf{Setup\text{-}ltr}(1^\lambda)$ and computes an obfuscation of $\mathsf{Prog\text{-}1}\{M_i, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{ltr}}, k', k_1, \ldots, k_\lambda, k_{\mathrm{sig},A}\}$. It sends this obfuscated program to $\mathcal{A}$ as the constrained key.

Finally, after all constrained key queries, $\mathcal{A}$ sends its guess b', which $\mathcal{B}$ forwards to the challenger. Note that if $\mathcal{A}$ wins the security game against PRF, then $\mathcal{B}$ wins the security game against F. This concludes our proof.

■

5 Attribute Based Encryption for Turing Machines

In this section, we describe an ABE scheme where policies are associated with Turing machines, and as a result, attributes can be strings of unbounded length. Our ABE scheme is very similar to the constrained PRF construction described in Section 4.

Let $\mathcal{PKE} = (\mathsf{PKE.setup}, \mathsf{PKE.enc}, \mathsf{PKE.dec})$ be a public key encryption scheme and F a puncturable PRF for Turing machines, with algorithms PRF.setup and PRF.constrain. Consider the following ABE scheme:

- $\mathsf{ABE.setup}(1^\lambda)$ The setup algorithm chooses a puncturable PRF key $k \leftarrow \mathsf{F.setup}(1^\lambda)$ and $(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. Next, it computes an obfuscation of $\mathsf{Prog\text{-}PK}\{k\}$ (defined in Figure 4). The public key $\mathrm{PK}_{\mathsf{ABE}} = (\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0, i\mathcal{O}(\mathsf{Prog\text{-}PK}\{k\}))$, while the master secret key is $\mathrm{MSK}_{\mathsf{ABE}} = k$.

Program Prog-PK

Constants: Puncturable PRF key k
Input: Accumulation of input acc-inp

1. Let $F(k, \mathsf{acc\text{-}inp}) = r$. Compute $(\mathrm{pk}, \mathrm{sk}) = \mathsf{PKE.setup}(1^\lambda; r)$.
2. Output pk.

Figure 4: Program Prog-PK

- $\mathsf{ABE.enc}(m, x, \mathrm{PK}_{\mathsf{ABE}})$ Let $\mathrm{PK}_{\mathsf{ABE}} = (\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0, \mathsf{Program}_{pk})$ and $x = x_1 \ldots x_n$. As in Section 4, the encryption algorithm first 'accumulates' the attribute x using the accumulator public parameters. Let $\mathsf{acc\text{-}inp} = \mathsf{acc}_n$, where for all $j \leq n$, acc_j is defined as follows:
  - $\mathrm{STORE}_j = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{j-1}, j-1, x_j)$
  - $aux_j = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{j-1}, j-1)$
  - $\mathsf{acc}_j = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_{j-1}, x_j, j-1, aux_j)$

  Next, the accumulated value is used to compute a PKE public key. Let $\mathrm{pk} = \mathsf{Program}_{pk}(\mathsf{acc\text{-}inp})$. Finally, the algorithm outputs $\mathsf{ct} = \mathsf{PKE.enc}(m, \mathrm{pk})$.

- $\mathsf{ABE.keygen}(\mathrm{MSK}_{\mathsf{ABE}}, M)$ Let $\mathrm{MSK}_{\mathsf{ABE}} = k$ and $M =$ a Turing machine. The ABE key corresponding to M is exactly the constrained key corresponding to M, as defined in Section 4. In particular, the key generation algorithm chooses $(\mathrm{PP}_{\mathsf{ltr}}, \mathsf{it}_0) \leftarrow \mathsf{Setup\text{-}ltr}(1^\lambda, T)$ and a puncturable PRF key $k_{\mathrm{sig},A}$, and computes an obfuscation of $\mathsf{Prog}\{M, k, k_{\mathrm{sig}}, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{ltr}}\}$ (defined in Figure 1) and $\mathsf{Init\text{-}Sign}\{k_{\mathrm{sig},A}$ (defined in Figure 2). The secret key $\mathrm{SK}\{M\} = (\mathrm{PP}_{\mathsf{ltr}}, \mathsf{it}_0, i\mathcal{O}(\mathsf{Prog}), i\mathcal{O}(\mathsf{Init\text{-}Sign}))$.

- $\mathsf{ABE.dec}(\mathrm{SK}\{M\}, \mathsf{ct}, x)$ Let $\mathrm{SK}\{M\} = (\mathrm{PP}_{\mathsf{ltr}}, \mathsf{it}_0, \mathsf{Program}_1, \mathsf{Program}_2)$, and suppose M accepts x in t^* steps. As in the constrained key PRF evaluation, the decryption algorithm first obtains a signature using $\mathsf{Program}_2$ and then runs $\mathsf{Program}_1$ for t^* steps, until it outputs the pseudorandom string r. Using this PRF output r, the decryption algorithm computes $(\mathrm{pk}, \mathrm{sk}) = \mathsf{PKE.setup}(1^\lambda; r)$ and then decrypts ct using sk. The algorithm outputs $\mathsf{PKE.dec}(\mathrm{sk}, \mathsf{ct})$.

5.1 Proof of Security

We will first define a sequence of hybrid experiments, and then show that any two consecutive hybrid experiments are computationally indistinguishable.

5.1.1 Sequence of Hybrid Experiments

Hybrid H_0 This corresponds to the selective security game. Let x^* denote the challenge input, and $\mathsf{acc\text{-}inp}^* = \mathsf{Hash\text{-}Acc}(x^*)$.

Hybrid H_1 In this hybrid, the challenger sends an obfuscation of Prog-1 instead of Prog. Prog-1, on inputs corresponding to $\mathsf{acc\text{-}inp}^*$, never reaches the accepting state q_{ac}. This is similar to Hybrid_1 of the constrained PRF security proof in Section 4.1.

Hybrid H_2 In this hybrid, the challenger first punctures the PRF key k at $\mathsf{acc\text{-}inp}^*$. It computes $k' \leftarrow \mathsf{F.puncture}(k, \mathsf{acc\text{-}inp}^*)$ and $(\mathrm{pk}^*, \mathrm{sk}^*) = \mathsf{PKE.setup}(1^\lambda; F(k, \mathsf{acc\text{-}inp}^*))$. Next, it uses k' and pk^* to define $\mathsf{Prog\text{-}PK}'\{k', \mathrm{pk}^*\}$ (see Figure 5). It sends an obfuscation of $\mathsf{Prog\text{-}PK}'$ as the public key. Next, for each of the secret key queries, it sends an obfuscation of Prog-1. However unlike the previous hybrid, Prog-1 has k' hardwired instead of k.

Program $\mathsf{Prog\text{-}PK}'$

Constants: Punctured PRF key k', Hardwired accumulation $\mathsf{acc\text{-}inp}^*$ and public key pk^*.
Input: Accumulation of input $\mathsf{acc\text{-}inp}$

1. If $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, set $\mathrm{pk} = \mathrm{pk}^*$.
   Else let $F(k', \mathsf{acc\text{-}inp}) = r$. Compute $(\mathrm{pk}, \mathrm{sk}) = \mathsf{PKE.setup}(1^\lambda; r)$.
2. Output pk.

Figure 5: Program $\mathsf{Prog\text{-}PK}'$

Hybrid H_3 In this hybrid, the challenger chooses $(\mathrm{pk}^*, \mathrm{sk}^*) \leftarrow \mathsf{PKE.setup}(1^\lambda)$; that is, the public key is computed using true randomness. It then hardwires pk^* in $\mathsf{Prog\text{-}PK}$. The secret key queries are same as in previous hybrids.

5.1.2 Analysis

Let $\mathsf{Adv}_i^{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in hybrid H_i.

Lemma 5.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, Acc is a secure positional accumulator, Itr is a secure iterator, $\mathcal{S}$ is a secure splittable signature scheme and F is a secure puncturable PRF, for any adversary $\mathcal{A}$, $|\mathsf{Adv}_0^{\mathcal{A}} - \mathsf{Adv}_1^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

The proof of this lemma is identical to the proof of Lemma 4.1.

Lemma 5.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Similar to the proof of Lemma 4.2, k can be replaced with k' in all the secret key queries, since $F(k, \mathsf{acc\text{-}inp}^*)$ is never executed. As far as $\mathsf{Prog\text{-}PK}$ and $\mathsf{Prog\text{-}PK}'$ are concerned, $(\mathrm{pk}^*, \mathrm{sk}^*)$ is set to be $\mathsf{PKE.setup}(1^\lambda; F(k, \mathsf{acc\text{-}inp}^*))$, and therefore, the programs are functionally identical. ∎

Lemma 5.3. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_2^{\mathcal{A}} - \mathsf{Adv}_3^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this follows immediately from the security definition of puncturable PRFs. Suppose there exists an adversary that can distinguish between H_2 and H_3 with advantage ϵ. Then, there exists a PPT algorithm $\mathcal{B}$ that can break the selective security of F. $\mathcal{B}$ first receives x^* from the adversary. It computes $\mathsf{acc\text{-}inp}^*$, sends $\mathsf{acc\text{-}inp}^*$ to the PRF challenger and receives k', y, where y is either the PRF evaluation at $\mathsf{acc\text{-}inp}^*$, or a truly random string. Using y, it computes $(\mathrm{pk}^*, \mathrm{sk}^*) = \mathsf{PKE.setup}(1^\lambda; y)$, and uses k', pk^* to define the public key $i\mathcal{O}(\mathsf{Prog\text{-}PK}'\{k', \mathrm{pk}^*\})$. The secret key queries are same in both hybrids, and can be answered using k' only. As a result, $\mathcal{B}$ simulates either H_2 or H_3 perfectly. This concludes our proof. ∎

Lemma 5.4. Assuming $\mathcal{PKE}$ is a secure public key encryption scheme, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_3^{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}_3^{\mathcal{A}} = \epsilon$. Then there exists a PPT adversary $\mathcal{B}$ that breaks IND-CPA security of $\mathcal{PKE}$. $\mathcal{B}$ receives a public key pk^* from the challenger. It chooses PRF key k, punctures it at $\mathsf{acc\text{-}inp}^*$ and sends the public key $i\mathcal{O}\{\mathsf{Prog\text{-}PK}'\}$. Next, it responds to the secret key queries, and finally, on receiving challenge messages m_0, m_1, it forwards them to the challenger, and receives ct^*, which it forwards to the adversary. The post challenge key query phase is also simulated perfectly, since it has all the required components. ∎

Acknowledgements. We would like to thank the anonymous reviewers for their valuable feedback and comments.

References

[ACC+15] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating ram computations with adaptive soundness and privacy. Cryptology ePrint Archive, Report 2015/1082, 2015. http://eprint.iacr.org/2015/1082.

[AF16] Hamza Abusalah and Georg Fuchsbauer. Constrained prfs for unbounded inputs with short keys. Cryptology ePrint Archive, Report 2016/279, 2016. http://eprint.iacr.org/.

[AFP14] Hamza Abusalah, Georg Fuchsbauer, and Krzysztof Pietrzak. Constrained prfs for unbounded inputs. *IACR Cryptology ePrint Archive*, 2014:840, 2014.

[AFP16] Hamza Abusalah, Georg Fuchsbauer, and Krzysztof Pietrzak. Constrained prfs for unbounded inputs. In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, pages 413–428, 2016.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 308–326, 2015.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, pages 125–153, 2016.

[BFP+15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 31–60, 2015.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519, 2014.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 439–448, 2015.

[BV15a] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 171–190, 2015.

[BV15b] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 1–30, 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, pages 280–300, 2013.

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

[CCHR15] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Succinct adaptive garbled ram. Cryptology ePrint Archive, Report 2015/1074, 2015. http://eprint.iacr.org/2015/1074.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 429–437, 2015.

[CRV14] Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Constrained pseudorandom functions: Verifiable and delegatable. Cryptology ePrint Archive, Report 2014/522, 2014. http://eprint.iacr.org/.

[DDM16] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Verifiable and delegatable constrained pseudorandom functions for unconstrained inputs. Cryptology ePrint Archive, Report 2016/784, 2016. http://eprint.iacr.org/2016/784.

[FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained prfs. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 82–101, 2014.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[GGH+13b] 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.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 518–535, 2014.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 151–170, 2015.

[HKKW14] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. *IACR Cryptology ePrint Archive*, 2014:720, 2014.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, pages 79–102, 2015.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 163–172, 2015.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 668–697, 2015.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 419–428, New York, NY, USA, 2015. ACM.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM Conference on Computer and Communications Security*, 2013.

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In *ASIACRYPT*, 2015.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[Zha14] Mark Zhandry. Adaptively secure broadcast encryption with small system parameters, 2014.

A Cryptographic Primitives Compatible with iO

A.1 The Need for iO-Compatible Primitives

To see why we require $i\mathcal{O}$-compatible cryptographic primitives, let us consider the following toy example. Suppose we have a security game between a challenger and an adversary, where the challenger first chooses a signing key SK and verification key VK of a secure signature scheme. Next, the challenger chooses a secret message s, and computes an obfuscation of the program P defined in Figure 6 below. The program takes as input a message m and a signature σ, and it outputs the secret s if σ is a valid signature corresponding to m. The challenger sends an obfuscation of P, along with VK to the adversary.

Program P

Constants: Secret s, Verification key VK

Input: Message m, signature σ

1. if $\mathsf{Verify}(\mathrm{VK}, m, \sigma) = 1$, output s. Else output $\perp$.

Figure 6: Program P

Intuitively, we'd expect that the adversary cannot learn the secret s, since the program is obfuscated, and the adversary cannot forge a signature. In particular, we want to use the security of the obfuscator and the signature scheme to say that the obfuscation of P is indistinguishable from the obfuscation of a circuit $P_\perp$ that outputs '$\perp$' on all inputs. This would imply that the adversary cannot learn the secret s. If the obfuscator was a differing input obfuscator, then one can claim that if an adversary can distinguish between an obfuscation of P and an obfuscation of $P_\perp$, then one can use this adversary to output a signature forgery.

However, using $i\mathcal{O}$ is tricky, and general signature schemes are not $i\mathcal{O}$-compatible. Therefore, we require some other properties of the signature scheme when using it with indistinguishability obfuscation. In the toy example above, we can prove security under $i\mathcal{O}$ by extending the signature scheme as follows. Let us assume the signature scheme has two modes for setup: a standard mode in which the setup algorithm outputs a signing key and a verification key as usual, and a rejection mode in which the setup outputs a 'rejection verification key' that does not accept any signature. Suppose this signature scheme has the property that no PPT adversary can distinguish between a verification key output by standard mode setup and a verification key output by rejection mode setup [9]. Then we can prove security in the toy example. The program P can be first modified to have a rejection verification key instead of the standard verification key. This is indistinguishable because of the (new) security property of the signature scheme. Next, we can replace the program P with $P_\perp$. This step is computationally indistinguishable due to the security of $i\mathcal{O}$, and due to the fact that if P has a rejection key, then it always outputs $\perp$.

B Proof of Lemma 1

In Lemma 1, we need to show that hybrid experiments $\mathsf{Hybrid}_{0,j}$ and $\mathsf{Hybrid}_{0,j+1}$ are computationally indistinguishable. The only difference between $\mathsf{Hybrid}_{0,j}$ and $\mathsf{Hybrid}_{0,j+1}$ corresponds to the $(j+1)^{th}$ query. In one case, the challenger sends an obfuscation of Prog_{j+1} [10], while in the other, it sends an obfuscation of $\mathsf{Prog\text{-}1}_{j+1}$ (the other components - $\mathsf{Init\text{-}Sign}$, $\mathrm{PP}_{\mathsf{ltr}}$ and it_0 are the same in both hybrids). To prove this lemma, we will define a sequence of intermediate hybrids to gradually transform Prog to $\mathsf{Prog\text{-}1}$.

Recall, t^*_{j+1} denotes the running time of M_{j+1} on input x^*, and τ^*_{j+1} denotes the smallest power of two greater than t^*_{j+1}. For simplicity of notation, in this section, we will skip the $(j+1)$ in subscript.

[9] Note that in this security game, the adversary is not given any signature queries. Clearly, if the adversary had signature queries, then it can easily distinguish between a standard verification key and a rejection mode verification key.

[10] Recall Prog_{j+1} refers to the program Prog with the $(j+1)^{th}$ query Turing machine M_{j+1} hardwired. Similarly, $\mathsf{Prog\text{-}1}_{j+1}$ is $\mathsf{Prog\text{-}1}$ with M_{j+1} hardwired.

As mentioned in the proof outline (Section 4.1), we will first define a pre-processing hybrid $\mathsf{Hyb}_{0,j}$-1 which introduces 'bad' valid signatures. Next, we gradually ensure that the program does not reach accepting state for inputs corresponding to $\mathsf{acc\text{-}inp}^*$ at time steps $t \leq i$. This is done in the hybrid $\mathsf{Hyb}_{(0,j)}$-$(1,i)$. In this hybrid, the program outputs a good signature at time step i for inputs corresponding to $\mathsf{acc\text{-}inp}^*$ only if the input symbol, iterator and accumulator are the correct ones. Next, we define a hybrid $\mathsf{Hyb}'_{(0,j)}$-$(1,i)$ which outputs a good signature at time step i for inputs corresponding to $\mathsf{acc\text{-}inp}^*$ only if the *output* symbol, iterator and accumulator are the correct ones. This acts as a bridge between $\mathsf{Hyb}_{(0,j)}$-$(1,i)$ and $\mathsf{Hyb}_{(0,j)}$-$(1,i+1)$. Further details can be found below.

Sequence of Hybrids

$\mathsf{Hyb}_{0,j}$-0 : This corresponds to $\mathsf{Hybrid}_{0,j}$. Let P_0 denote Prog_{j+1}.

$\mathsf{Hyb}_{0,j}$-1 : In this hybrid, the challenger outputs an obfuscation of program P_1 (defined in Figure 7) instead of P_0. To define this program, the challenger first chooses another PRF key $k_{\mathrm{sig},B}$ to sign/verify 'B' type signatures. Using $k_{\mathrm{sig},B}$, it defines the program $P_1 \equiv P_1\{M_{j+1}, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k, k_{\mathrm{sig},A}, k_{\mathrm{sig},B}\}$. The $(j+1)^{th}$ constrained key query response is $K\{M_{j+1}\} = (\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \textsc{store}_0, \mathrm{PP}_{\mathsf{Itr}}, \mathsf{it}_0, iO(P_1), i\mathcal{O}(\mathsf{Init\text{-}Sign}))$.

We will now define $2t^*$ hybrid experiments $\mathsf{Hyb}_{0,j}$-$(1,i)$ and $\mathsf{Hyb}'_{0,j}$-$(1,i)$ for $0 \leq i \leq t^*$.

$\mathsf{Hyb}_{0,j}$-$(1,i)$ In this hybrid, the challenger first computes the 'correct message' m_i to be signed at step i. The message m_i is computed as follows:

Let $\mathsf{st}_0 = q_0$, $\mathsf{pos}_0 = 0$. For $j = 1$ to i:

1. $(\mathsf{sym}_j, \pi_j) = \mathsf{Prep\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, \textsc{store}_{j-1}, \mathsf{pos}_{j-1})$.
2. $aux_j = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, \textsc{store}_{j-1}, \mathsf{pos}_{j-1})$.
3. $(\mathsf{st}_j, \mathsf{sym}_{w,j}, \beta) = \delta(\mathsf{st}_{j-1}, \mathsf{sym}_{j-1})$.
4. $\mathsf{acc}_j = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_{j-1}, \mathsf{sym}_{w,j}, \mathsf{pos}_{j-1}, aux_j)$.
5. $\mathsf{it}_j = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, \mathsf{it}_{j-1}, (\mathsf{st}_{j-1}, \mathsf{acc}_{j-1}, \mathsf{pos}_{j-1}))$.
6. $store_j = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, \textsc{store}_{j-1}, \mathsf{pos}_{j-1}, \mathsf{sym}_{w,j})$.
7. $\mathsf{pos}_j = \mathsf{pos}_{j-1} + \beta$.

It sets $m_i = (\mathsf{it}_i, \mathsf{st}_i, \mathsf{acc}_i, \mathsf{pos}_i)$ and computes the obfuscation of $P_{1,i} \equiv P_{1,i}\{\mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k, k_{\mathrm{sig},A}, k_{\mathrm{sig},B}, m_i\}$ (defined in Figure 8). The remaining components of the constrained key are same as in H_1.

$\mathsf{Hyb}'_{0,j}$-$(1,i)$: As in the previous hybrid, the challenger first computes the 'correct message' m_i to be signed at step i. Next, it computes an obfuscation of $P'_{1,i} \equiv P'_{1,i}\{\mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k, k_{\mathrm{sig},A}, k_{\mathrm{sig},B}, m_i\}$ (defined in Figure 9).

$\mathsf{Hyb}_{0,j}$-2 In this hybrid, the challenger outputs an obfuscation of P_2 (defined in Figure 10) which is functionally identical to P'_{1,t^*-1}.

Program P_1

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$.
   (c) Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$
   (d) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (e) If $\alpha =$'-' and $(t > t^*$ **or** $t \leq 1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*)$, output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (f) If $\alpha =$'-' output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$.
   (b) Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (c) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 7: Program P_1

Analysis Let $\mathsf{Adv}^{\mathcal{A}}_y$ denote the advantage of adversary $\mathcal{A}$ in hybrid experiment $\mathsf{Hyb}_{0,j}$-y, and $\mathsf{Adv}'^{\mathcal{A}}_y$ the advantage of $\mathcal{A}$ in hybrid experiment $\mathsf{Hyb}'_{0,j}$-y. The proofs of Lemma B.1, B.2, B.3, B.4 and B.5 are along the same lines as the corresponding proofs in [KLW15].

Program $P_{1,i}$

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$
Hardwired accumulated input $\mathsf{acc\text{-}inp}^*$, correct message m_i

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.

   (c) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (d) If $\alpha =$'-' and $(t > t^*$ **or** $t \leq i$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*)$, output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (e) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   (b) Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (c) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   (d) If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$, $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 8: Program $P_{1,i}$

Program $P'_{1,i}$

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$
Hardwired accumulated input $\mathsf{acc\text{-}inp}^*$, correct message m_i

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.

4. (a) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (b) If $\alpha =$'-' and $(t > t^*$ **or** $t \leq i+1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*)$, output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (c) If $\alpha =$'-' output $\perp$.

5. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i+1$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

6. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

7. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   (b) Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (c) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$, $m_{\text{in}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

8. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \text{␣}$.
9. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 9: Program $P'_{1,i}$

Program P_2

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$
Hardwired accumulated input $\mathsf{acc\text{-}inp}^*$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (c) Let $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   (b) Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (c) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, t^*)$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 10: Program P_2

Lemma B.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure pseudorandom function, $\mathcal{S}$ is a secure splittable signature scheme satisfying Definition 2.2 , for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_0^{\mathcal{A}} - \mathsf{Adv}_1^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

The proof of this lemma is described in Appendix B.1.

Next, we will show that $\mathsf{Hyb}_{0,j}$-1 and $\mathsf{Hyb}'_{0,j}$-$(1,0)$ are computationally indistinguishable.

Lemma B.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure pseudorandom function, $\mathcal{S}$ is a secure splittable signature scheme satisfying Definition 2.3 and Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}'^{\mathcal{A}}_{1,0}| \leq \text{negl}(\lambda)$.

Program P_3

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$
Hardwired accumulated input $\mathsf{acc\text{-}inp}^*$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $t > t^*$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, output $\perp$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
4. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (c) Let $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
5. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
6. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
7. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   (b) Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (c) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, t^*)$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
8. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \text{␣}$.
9. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 11: Program P_3

The proof of this lemma is given in Appendix B.2.

Lemma B.3. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure pseudorandom function, $\mathcal{S}$ is a secure splittable signature scheme satisfying Definitions 2.3, 2.4, 2.5 and Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{1,i} - \mathsf{Adv}'^{\mathcal{A}}_{1,i}| \leq \text{negl}(\lambda)$.

The proof of this lemma is described in Appendix B.3.

Lemma B.4. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure pseudorandom function, Acc satisfies indistinguishability of Read Setup (Definition 2.6) and indistinguishability of Write Setup (Definition 2.7) and Itr satisfies indistinguishability of Setup (Definition 2.10), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}'^{\mathcal{A}}_{1,i} - \mathsf{Adv}^{\mathcal{A}}_{1,i+1}| \leq \mathrm{negl}(\lambda)$.

The proof of this lemma is described in Appendix B.4.

Lemma B.5. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and Acc satisfies indistinguishability of Read Setup (Definition 2.6) , for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}'^{\mathcal{A}}_{1,t^*-1} - \mathsf{Adv}^{\mathcal{A}}_{2}| \leq \mathrm{negl}(\lambda)$.

The proof of this lemma is described in Appendix B.5.

Lemma B.6. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure pseudorandom function and PRG is a secure pseudorandom generator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{2} - \mathsf{Adv}^{\mathcal{A}}_{3}| \leq \mathrm{negl}(\lambda)$.

The proof of this lemma involves handling the 'tail hybrid'. As mentioned in Section 1, our approach for handling this hybrid is different from the one in [KLW15]. This is because the [KLW15] approach leads to exponential intermediate hybrids, while in our case, the security loss is only polynomial in the security parameter. The proof is described in Appendix B.6.

B.1 Proof of Lemma B.1

Proof Intuition Let us consider the differences between P_0 and P_1.

1. For inputs corresponding to $t > t^*$ or $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$, both programs are identical.
2. For inputs corresponding to $t \leq t^*$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, P_1 first checks if it is an 'A' type signature. If not, it checks if it a 'B' type signature. The output signature (if any) is of the same type as the input signature. Also, if the incoming signature is a 'B' type signature and $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{ac}}$, then the program cannot output $\mathsf{F}(k, \mathsf{acc\text{-}inp}^*)$; instead, it aborts.

So, we need to allow 'B' type signatures for steps $1 \leq t \leq t^*$ if $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. We do this in a 'top-down' manner, and define intermediate hybrid experiments $H_{t^*}, \ldots, H_0$, where H_i outputs P_0-i, which allows only 'A' type signatures for $t \leq i$ or $t > t^*$ if $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$.

Consider the programs P_0-i and P_0-$(i-1)$: the only difference is corresponding to inputs with $t = i$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. On such inputs, P_0-i accepts only 'A' type signatures, while P_0-$(i-1)$ accepts both 'A' and 'B' type. We modify P_0-i as follows: on inputs corresponding to $t = i$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, it verifies using VK_A, and if this fails, it verifies using $\mathrm{VK}_{B,\mathrm{rej}}$. Clearly, if VK_A verification fails, then the program outputs $\perp$ since $\mathrm{VK}_{B,\mathrm{rej}}$ verification always fails. Next, we replace $\mathrm{VK}_{B,\mathrm{rej}}$ with VK_B, therefore allowing 'B' type signatures to be verified. Here, our 'top-down' approach is crucial - to replace $\mathrm{VK}_{B,\mathrm{rej}}$ with VK_B, it is important that the program does not output any B type signatures corresponding to $t = i$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, which is the case here. As a result, we can show that P_0-i and P_0-$(i-1)$ are computationally indistinguishable. Note that there will also be additional hybrids, where we puncture/unpuncture the PRF key used to generate the 'B' type verification keys.

Formal proof We will first define t^* intermediate hybrids H_0-$1, \ldots, H_0$-t^* such that H_0-t^* corresponds to Hyb_0 and H_0-1 corresponds to Hyb_1.

Hybrid H_0-i In this experiment, the challenger outputs an obfuscation of P_0-$i\{M_{j+1}, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k, k_{\mathrm{sig},A}, k_{\mathrm{sig},B}\}$ (defined in Figure 12).

In order to show that Hyb_0 and Hyb_1 are computationally indistinguishable, it suffices to show that H_0-i and H_0-$(i-1)$ are computationally indistinguishable. Let $\mathsf{Adv}^i_{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in H_i.

Claim B.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a secure puncturable PRF and $\mathcal{S}$ is a splittable signature scheme satisfying Definition 2.2, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^i_{\mathcal{A}} - \mathsf{Adv}^{i-1}_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. We will first define intermediate hybrid experiments H_0-i-$0, \ldots, H_0$-i-5.

Program P_0-i

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (c) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (d) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (e) If $\alpha =$'-' output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 12: Program P_0-i

Hybrid H_0-i-0 This corresponds to H_0-i.

Hybrid H_0-i-1 In this hybrid, the challenger outputs an obfuscation of P_0-i-1$\{M_{j+1}, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, k\ k_{\text{sig},A}, k_{\text{sig},B}\}$ (described in Figure 13).

Program P_0-i-1

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (c) Set $\text{VK} = \text{VK}_{B,\text{rej}}$.
   (d) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (e) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i-1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $t = i$ and $\mathsf{Verify\text{-}Spl}(\text{VK}, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (f) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 13: Program P_0-i-1

Hybrid H_0-i-2 In this hybrid, the challenger first punctures the PRF key $k_{\text{sig},B}$ on input $(\mathsf{acc\text{-}inp}^*, i-1)$. It computes $k'_{\text{sig},B} \leftarrow \mathsf{F.puncture}(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, i-1))$. Next, it computes $r_C = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, i-1))$ and $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$. It hardwires $k'_{\text{sig},B}$ and $\text{VK}_{C,\text{rej}}$ in the program P_0-i-2$\{M$, PP_{Acc}, PP_{Itr}, k, $k_{\text{sig},A}$, $k_{\text{sig},B}$, $\text{VK}_{C,\text{rej}}\}$ (defined in Figure 14).

P_0-i-2

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}$, punctured PRF key $k'_{\text{sig},B}$
Hardwired verification key VK_C

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   (b) Let $r_{\text{sig},B} = F(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (c) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (d) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i-1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $t = i$ and $\mathsf{Verify\text{-}Spl}(\text{VK}_C, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (e) If $\alpha =$'-' output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 14: P_0-i-2

Hybrid H_0-i-3 This experiment is similar to $H_{i,b}$, except that r_C is chosen uniformly at random. More formally, the challenger computes punctured key $k'_{\text{sig},B}$ as before. However, it chooses $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$. The obfuscated program has $\text{VK}_{C,\text{rej}}$ hardwired as before.

Hybrid H_0-i-4 In this hybrid, the challenger chooses $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ as before. However, instead of hardwiring $\mathrm{VK}_{C,\mathrm{rej}}$, it hardwires VK_C.

Hybrid H_0-i-5 In this hybrid, the challenger uses a pseudorandom string to compute $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}})$. More formally, the challenger computes $r_C = F(k_{\mathrm{sig},B}, (\mathsf{acc\text{-}inp}^*, i-1))$, $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$. It hardwires VK_C.

Hybrid H_0-i-6 This experiment corresponds to H_0-$(i-1)$.

Analysis Let $\mathsf{Adv}^y_{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in H_0-i-y.

Claim B.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT $\mathcal{A}$, $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. First, since $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{B,\mathrm{rej}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 0$ for all $m_{\mathrm{in}}, \sigma_{\mathrm{in}}$, both programs output $\perp$ when $\alpha =$'B' and $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$. For inputs corresponding to $t \neq i$ or $t > t^*$ or $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$, both programs have same functionality. Therefore, both programs have identical functionality, and their obfuscations are computationally indistinguishable. ∎

Claim B.3. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT $\mathcal{A}$, $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The only difference between P_0-i-1 and P_0-i-2 is that the latter uses a punctured PRF key $k'_{\mathrm{sig},B}$ which can evaluate the PRF at all points except $(\mathsf{acc\text{-}inp}^*, i-1)$. During the verification phase, the PRF generated 'B' type verification key is not used if $t = i$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. During the signing phase, the signing algorithm needs to compute 'B' type keys only for $t \geq i$. As a result, $k'_{\mathrm{sig},B}$ is not evaluated on input $(\mathsf{acc\text{-}inp}^*, i-1)$. This ensures that the two programs are functionally identical. ∎

Claim B.4. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. If there exists an adversary $\mathcal{A}$ such that $\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}} = \epsilon$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the puncturable PRF security. It receives a punctured key $k'_{\mathrm{sig},B}$ and z, which is either a truly random string or the pseudorandom evaluation at $(\mathsf{acc\text{-}inp}^*, i-1)$. $\mathcal{B}$ then chooses the remaining components by itself, and using $k'_{\mathrm{sig},B}$ and z, simulates either H_0-i-2 or H_0-i-3. ∎

Claim B.5. Assuming $\mathcal{S}$ is a splittable signature scheme satisfying $\mathrm{VK}_{\mathrm{rej}}$ indistinguishability (Definition 2.2), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Here, we rely crucially on the fact that SK_C was not hardwired in the program. As a result, given only VK_C or $\mathrm{VK}_{C,\mathrm{rej}}$, the experiments are indistinguishable. ∎

Claim B.6. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^4_{\mathcal{A}} - \mathsf{Adv}^5_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This step is similar to the proof of Claim B.4, and follows analogously from the security of the puncturable PRF. ∎

Claim B.7. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT $\mathcal{A}$, $|\mathsf{Adv}^5_{\mathcal{A}} - \mathsf{Adv}^6_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The only difference between H_0-i-5 and H_0-i-6 is that H_0-i-5 uses a PRF key $k'_{\text{sig},B}$ punctured at $(\mathsf{acc\text{-}inp}^*, i-1)$, while H_0-1-6 uses $k_{\text{sig},B}$ itself. Using the correctness property of puncturable PRFs, we can argue that the programs output by these two hybrids are functionally identical, and therefore, by $i\mathcal{O}$, the hybrids are computationally indistinguishable. ∎

To conclude, for any PPT adversary $\mathcal{A}$, if $\mathcal{A}$ has advantage $\mathsf{Adv}^0_{\mathcal{A}}$ in Hybrid_0 and $\mathsf{Adv}^1_{\mathcal{A}}$ in Hybrid_1, then $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

∎

B.2 Proof of Lemma B.2

Proof Intuition: In this lemma, we switch from Hyb_1 to $\mathsf{Hyb}'_{1,0}$. In Hyb_1, the challenger outputs an obfuscation of P_1 (see Figure 7), while in $\mathsf{Hyb}'_{1,0}$, the challenger outputs an obfuscation of $P'_{1,0}$ (see Figure 9). There are two major differences between these programs. First, $P'_{1,0}$ aborts if the output state is accepting at $t = 1$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$. Secondly, at $t = 1$ for inputs corresponding to $\mathsf{acc\text{-}inp}^*$, the program outputs a good signature only if the input accumulator, iterator and state are the correct ones. Intuitively, we can make these switches because the adversary only gets a good signature for $t = 0, \mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ (the $\mathsf{Init\text{-}Sign}$ program outputs this good signature). To begin with, the program P_1 is modified so that for $t = 1, \mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, the input accumulator, iterator and state must be correct. This is enforced using the splittable signatures properties. Next, once it is ensured that the accumulator is correct, we can be sure that the program does not reach accepting state. This is done using the read enforcing property of the accumulator.

Formal Proof: We will define a sequence of hybrids to prove this lemma.

B.2.1 Sequence of Hybrids

Hybrid H_1-0 This corresponds to Hyb_1.

Hybrid H_1-1 In this hybrid experiment, the challenger first punctures the PRF key $k_{\text{sig},A}$ at input $(\mathsf{acc\text{-}inp}^*, 0)$. It computes $k'_{\text{sig},A} \leftarrow \mathsf{F.puncture}(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, 0))$. Next, it computes the signing/verification keys generated using $r_C = \mathsf{F}(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, 0))$. It computes $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$ and $\sigma_C = \mathsf{Sign\text{-}Spl}(\text{SK}_C, (\mathsf{it}_0, q_0, \mathsf{acc\text{-}inp}^*, 0))$. It modifies the programs $\mathsf{Init\text{-}Sign}$ and Prog-1 to use the punctured key, and has the signing/verification keys hardwired. More formally, it defines programs $\mathsf{Init\text{-}Sign}' = \mathsf{Init\text{-}Sign}\{k'_{\text{sig},A}, \sigma_C\}$ (see Figure 16), $W_1 = P_1\text{-}1\{k, k'_{\text{sig},A}, k_{\text{sig},B}, \text{VK}_C, \mathsf{acc\text{-}inp}^*\}$ (see Figure 15). It computes $i\mathcal{O}(\mathsf{Init\text{-}Sign}')$ and $i\mathcal{O}(P_1\text{-}1)$, and outputs these as part of the constrained key query for machine M_{j+1}.

Hybrid H_1-2 In this hybrid, the challenger uses a truly random string to compute VK_C. It chooses $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and uses VK_C in the obfuscated program $P_1\text{-}1\{k, k'_{\text{sig},A}, k_{\text{sig},B}, \text{VK}_C, \mathsf{acc\text{-}inp}^*\}$.

Hybrid H_1-3 In this hybrid, the challenger 'punctures' VK_C at $m_0 = (\mathsf{it}_0, q_0, \mathsf{acc\text{-}inp}^*, 0)$. It chooses $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and then computes $(\sigma_C, \text{VK}_{C,\text{one}}, \text{SK}_{C,\text{abo}}, \text{VK}_{C,\text{abo}}) \leftarrow \mathsf{Split}(\text{SK}_C, m_0)$. It hardwires $\text{VK}_{C,\text{one}}$ instead of VK_C.

Hybrid H_1-4 In this hybrid, the challenger makes the accumulator 'read-enforcing' at position 0. The challenger computes $(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, 0)$. This ensures that the adversary cannot input the wrong symbol at time step 1 and accumulated input $\mathsf{acc\text{-}inp}^*$.

Program P_1-1

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},B}$, punctured PRF key $k'_{\text{sig},A}$
hardwired accumulated value $\mathsf{acc\text{-}inp}^*$, hardwired verification key VK_C

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, 1)$, let $r_{\text{sig},A} = F(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Else set $\text{VK}_A = \text{VK}_C$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (c) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (d) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq 1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (e) If $\alpha =$'-' output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_{\text{sig},A} = F(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 15: Program P_1-1

Hybrid H_1-5 Recall in the previous hybrid, the challenger made the accumulator 'read-enforcing' at position 0. As a result, the adversary's (state, symbol) input with $t = 1$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ must be (q_0, x_1).

Program Init-Sign′

Constants: Punctured PRF key $k'_{\mathrm{sig},A}$, Initial TM state q_0, Iterator value it_0, hardwired signature σ_C, hardwired accumulated value $\mathsf{acc\text{-}inp}^*$,

Input: Accumulation of input $\mathsf{acc\text{-}inp}$

1. If $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$, output σ_C.
   Else let $F(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}, 0)) = r_{\mathrm{sig},A}$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\mathrm{sig}})$. Output $\sigma = \mathsf{Sign\text{-}Spl}(\mathrm{SK}, (\mathsf{it}_0, q_0, \mathsf{acc\text{-}inp}, 0))$.

Figure 16: Program Init-Sign′

Therefore, in this experiment, the challenger outputs an obfuscation of the program $W_5 = P_1\text{-}2\{\mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k'_{\mathrm{sig},A}, k_{\mathrm{sig},B}, \mathrm{VK}_{C,\mathrm{one}}, m_0\}$ defined in Figure 17.

Hybrid H_1-6 In this hybrid, the challenger uses normal accumulator setup instead of read-enforced setup.

Hybrid H_1-7 In this hybrid, the challenger hardwires VK_C instead of $\mathrm{VK}_{C,\mathrm{one}}$ in the program P_1-2.

Hybrid H_1-8 In this hybrid, the challenger chooses $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}})$ using the pseudorandom string $r_C = \mathsf{F}(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, 0))$. It computes $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$.

Hybrid H_1-9 This corresponds to Hybrid $\mathsf{Hyb}'_{1,0}$.

B.2.2 Analysis

Let $\mathsf{Adv}_y^{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in hybrid H_1-y. We will now show that any PPT adversary has at most negligible advantage in consecutive hybrids.

Lemma B.7. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_0^{\mathcal{A}} - \mathsf{Adv}_1^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this lemma follows directly from the correctness of puncturable PRFs. Note that we need to show that Init-Sign and Init-Sign′ are functionally identical, and same for P_1 and P_1-1. For both these pairs of programs, program equivalence is immediate, since the punctured PRF output is correct at all non-punctured points. At the points of puncturing, the correct signing/verification keys are hardwired in the program. ∎

Lemma B.8. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this lemma follows from the selective security definition of puncturable PRFs. In Hybrid H_1-0, the challenger uses a pseudorandom string to compute $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}})$, while in Hybrid H_1-1, it uses a truly random string.

∎

Lemma B.9. Assuming $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{one}}$ indistinguishability (Definition 2.3), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_2^{\mathcal{A}} - \mathsf{Adv}_3^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}_2^{\mathcal{A}} - \mathsf{Adv}_3^{\mathcal{A}}| = \epsilon$. Then we can construct a reduction algorithm $\mathcal{B}$ that breaks the $\mathrm{VK}_{\mathrm{one}}$ indistinguishability of $\mathcal{S}$. $\mathcal{B}$ sends m_i to the challenger. The challenger chooses $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$, $(\sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{C,\mathrm{abo}}, \mathrm{VK}_{C,\mathrm{abo}})$ and $\mathcal{B}$ receives (σ, VK), where $\sigma = \sigma_{C,\mathrm{one}}$ and $\mathrm{VK} = \mathrm{VK}_C$ or $\mathrm{VK}_{C,\mathrm{one}}$. It chooses the remaining components (including $\mathrm{SK}_{D,\mathrm{abo}}$

and VK_D), and computes P_1-1$\{k'_{\mathrm{sig},A}, k_{\mathrm{sig},B}, \sigma, \mathrm{VK}, \mathrm{SK}_{D,\mathrm{abo}}, \mathrm{VK}_D\}$. Now, note that $\mathcal{B}$ perfectly simulates either H_1-2 or H_1-3, depending on whether VK was VK_C or $\mathrm{VK}_{C,\mathrm{one}}$. ∎

Lemma B.10. Assuming Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_3^{\mathcal{A}} - \mathsf{Adv}_4^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This follows directly from the 'Indistinguishability of Read Setup' security property (Definition 2.6) of positional accumulators. ∎

Lemma B.11. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_4^{\mathcal{A}} - \mathsf{Adv}_5^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The only inputs for which the two programs can possibly differ correspond to $(\mathsf{acc\text{-}inp}^*, 1)$. Also note that the accumulator is enforcing, and as mentioned in the hybrid description, the (state/symbol) input must be (q_0, x_1). Hence $\mathsf{st}_{\mathrm{out}}$ cannot be q_{ac}. Also, note that the verification key hardwired is $\mathrm{VK}_{C,\mathrm{one}}$, which only accepts $m_{\mathrm{in}} = m_0$. This ensures that if $m_{\mathrm{in}} = m_0$, $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, 0)$, then both programs output an 'A' type signature, else both output $\perp$. This concludes our proof. ∎

Lemma B.12. Assuming Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_5^{\mathcal{A}} - \mathsf{Adv}_6^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This follows directly from the 'Indistinguishability of Read Setup' security property (Definition 2.6) of positional accumulators. ∎

Lemma B.13. Assuming $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{one}}$ indistinguishability (Definition 2.3), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_6^{\mathcal{A}} - \mathsf{Adv}_7^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This follows directly from the '$\mathrm{VK}_{\mathrm{one}}$ indistinguishability' security property (Definition 2.3) of splittable signatures. ∎

Lemma B.14. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_7^{\mathcal{A}} - \mathsf{Adv}_8^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this lemma follows from the selective security definition of puncturable PRFs. In Hybrid H_1-7, the challenger uses a truly random string to compute $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}})$, while in Hybrid H_1-8, it uses a pseudorandom string.

∎

Lemma B.15. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_8^{\mathcal{A}} - \mathsf{Adv}_9^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. We need to argue that the corresponding programs as used in Hybrid H_1-8 and Hybrid H_1-9 are functionally identical. For $\mathsf{Init\text{-}Sign}$ and $\mathsf{Init\text{-}Sign}'$, both are functionally identical since σ_C is correctly computed using $\mathsf{F}(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, 0))$. Similarly, since VK_C is correctly computed using $\mathsf{F}(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, 0))$, both programs in P_1-2 and $P'_{1,0}$ are functionally identical. ∎

Program P_1-2

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},B}$, punctured PRF key $k'_{\text{sig},A}$
hardwired accumulated value $\mathsf{acc\text{-}inp}^*$, hardwired verification key VK_C

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, 1)$, let $r_{\text{sig},A} = F(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Else set $\text{VK}_A = \text{VK}_C$.
   (b) Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (c) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (d) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq 1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (e) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq 1$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) Let $r'_{\text{sig},A} = F(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, 1)$ and $m_{\text{in}} = m_0$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, 1)$, $m_{\text{in}} \neq m_0$ output $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 17: Program P_1-2

B.3 Proof of Lemma B.3

Proof Intuition Let us first note the differences between $P_{1,i}$ and $P'_{1,i}$.

Input corr. to	$P_{1,i}$	$P'_{1,i}$
$t > t^*$ or $t < i$ or $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$	Verify 'A' signatures only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature.	Verify 'A' signatures only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature.
$t = i$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$	Verify 'A' signature only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature if $m_{\text{out}} = m_i$, 'B' signature if $m_{\text{out}} \neq m_i$.	Verify 'A' signatures only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature.
$t = i+1$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$	Verify 'A/B' signatures, output $\perp$ if 'B' type signature and $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output signature of same type as incoming signature.	Verify 'A' signature only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature if $m_{\text{in}} = m_i$, 'B' signature if $m_{\text{in}} \neq m_i$.
$i+2 \leq t \leq t^*$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$	Verify 'A/B' signatures, output $\perp$ if 'B' type signature and $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output signature of same type as incoming signature.	Verify 'A/B' signatures, output $\perp$ if 'B' type signature and $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output signature of same type as incoming signature.

As is evident from the table above, the only difference between the two programs is corresponding to $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t = i, i+1$. Program $P_{1,i}$ outputs a good signature at time $t = i$ for $\mathsf{acc\text{-}inp}^*$ only if the output accumulator, iterator and state are the correct ones. Program $P'_{1,i}$ aborts at $t = i+1, \mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ if the program reaches accepting state. If the state is not accepting, it outputs a good signature at time only if the input accumulator, iterator and state are the correct ones.

The first step is similar to the proof of Lemma B.2. Using the properties of splittable signatures, we modify $P_{1,i}$ so that, at time $t = i+1$, for $\mathsf{acc\text{-}inp}^*$, the only signature accepted is for the correct accumulator, iterator and state. This ensures that the input accumulator, iterator and state at $t = i+1$ for $\mathsf{acc\text{-}inp}^*$ are the correct ones. Next, using the read enforcing property of accumulator, we can ensure that the symbol input at $t = i+1$ is the correct one. Since both the input state and symbol are correct, the program cannot reach accepting state at $t = i+1$.

Formal Proof To prove Lemma B.3, we will first define a sequence of hybrids $H_0, \ldots, H_{13}$, where H_0 corresponds to $\mathsf{Hyb}_{0,j}(1, i)$ and H_{13} corresponds to $\mathsf{Hyb}'_{0,j}(1, i)$.

B.3.1 Sequence of Hybrid Experiments

Hybrid H_0 This experiment corresponds to $\mathsf{Hyb}_{0,j}(1, i)$.

Hybrid H_1 This experiment is identical to the previous one, except that the challenger guesses the position read by machine M_{j+1} on input x^* at step i. The challenger chooses $\mathsf{pos}^* \leftarrow [t^*]$ at the start of the experiment. The adversary then sends the challenge input x^*, receives string y^* and queries for constrained keys. On the $j+1^{th}$ query, the challenger checks if M_{j+1}, on input x^*, reads position pos^* at step i. If not, the challenger aborts, else it continues as in the previous hybrid.

Looking ahead, this guess will only be required for making the accumulator enforcing at position pos^*. However, we present this at the beginning of our hybrid sequence to simplify our analysis.

Hybrid H_2 In this experiment, the challenger punctures key $k_{\text{sig},A}, k_{\text{sig},B}$ at input $(\mathsf{acc\text{-}inp}^*, i)$, uses $F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$ and $F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, i))$ to compute $(\text{SK}_C, \text{VK}_C)$ and $(\text{SK}_D, \text{VK}_D)$ respectively. More formally, it computes $k'_{\text{sig},A} \leftarrow \mathsf{F.puncture}(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$, $r_C = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$, $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$ and $k'_{\text{sig},B} \leftarrow \mathsf{F.puncture}(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, i))$, $r_D = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, i))$, $(\text{SK}_D, \text{VK}_D, \text{VK}_{D,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_D)$.

It then hardwires $k'_{\text{sig},A}, k'_{\text{sig},B}$, $\text{SK}_C, \text{VK}_C, \text{SK}_D, \text{VK}_D$ in an altered program $W_1 = P_{1,i}\text{-}1\{k'_{\text{sig},A}, k'_{\text{sig},B}, \text{SK}_C, \text{VK}_C, \text{SK}_D, \text{VK}_D, m_i\}$(defined in Figure 18) and outputs its obfuscation. $P_{1,i}$-1 is identical to $P_{1,i}$, except that it uses a punctured PRF key $k'_{\text{sig},A}$ instead of $k_{\text{sig},A}$, and $k'_{\text{sig},B}$ instead of $k_{\text{sig},B}$. On input corresponding to $(\mathsf{acc\text{-}inp}^*, i)$, P uses the hardwired signing/verification keys.

Hybrid H_3 In this hybrid, the challenger chooses r_C, r_D uniformly at random instead of computing them using $F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$ and $F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, i))$. In other words, the secret key/verification key pairs are sampled as $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and $(\text{SK}_D, \text{VK}_D, \text{VK}_{D,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$.

Hybrid H_4 In this hybrid, the challenger computes constrained signing keys using the Split algorithm. As in the previous hybrids, it first computes the i^{th} message m_i. Then, it computes $(\sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \sigma_{C,\text{abo}}, \text{VK}_{C,\text{abo}}) = \mathsf{Split}(\text{SK}_C, m_i)$ and $(\sigma_{D,\text{one}}, \text{VK}_{D,\text{one}}, \sigma_{D,\text{abo}}, \text{VK}_{D,\text{abo}}) = \mathsf{Split}(\text{SK}_D, m_i)$.

It then hardwires $\sigma_{C,\text{one}}, \text{SK}_{D,\text{abo}}$ in $W_3 = P_{1,i}\text{-}2\{k'_{\text{sig},A}, k'_{\text{sig},B}, \sigma_{C,\text{one}}, \text{VK}_C, \text{SK}_{D,\text{abo}}, \text{VK}_D, m_i\}$ (defined in Figure 19) and outputs an obfuscation of P. Note that the only difference between $P_{1,i}$-2 and $P_{1,i}$-1 is that $P_{1,i}$-1, on input corresponding to step i, signs the outgoing message m using SK_C if $m = m_i$, else it signs using SK_D. On the other hand, at step i, $P_{1,i}$-2 outputs $\sigma_{C,\text{one}}$ if the outgoing message $m = m_i$, else it signs using $\text{SK}_{D,\text{abo}}$.

Hybrid H_5 This hybrid is similar to the previous one, except that the challenger hardwires $\text{VK}_{C,\text{one}}$ in $P_{1,i}$-2 instead of VK_C; that is, it computes $(\sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \sigma_{C,\text{abo}}, \text{VK}_{C,\text{abo}}) = \mathsf{Split}(\text{SK}_C, m_i)$ and $(\sigma_{D,\text{one}}, \text{VK}_{D,\text{one}}, \sigma_{D,\text{abo}}, \text{VK}_{D,\text{abo}}) = \mathsf{Split}(\text{SK}_D, m_i)$ and outputs an obfuscation of $W_4 = P_{1,i}\text{-}2\{k'_{\text{sig},A}, k'_{\text{sig},B}, \sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \text{SK}_{D,\text{abo}}, \text{VK}_D, m_i\})$.

Hybrid H_6 In this hybrid, the challenger hardwires $\text{VK}_{D,\text{abo}}$ instead of VK_D. As in the previous hybrid, it uses Split to compute $(\sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \sigma_{C,\text{abo}}, \text{VK}_{C,\text{abo}})$ and $(\sigma_{D,\text{one}}, \text{VK}_{D,\text{one}}, \sigma_{D,\text{abo}}, \text{VK}_{D,\text{abo}})$ from SK_C and SK_D respectively. However, it outputs an obfuscation of $W_5 = P_{1,i}\text{-}2\{k'_{\text{sig},A}, k'_{\text{sig},B}, \sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \text{SK}_{D,\text{abo}}, \text{VK}_{D,\text{abo}}, m_i\}$.

Hybrid H_7 In this hybrid, the challenger outputs an obfuscation of $P = P_{1,i}\text{-}3\{k'_{\text{sig},A}, k'_{\text{sig},B}, \sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \text{SK}_{C,\text{abo}}, \text{VK}_{C,\text{abo}}, m_i\}$ (described in Figure 20). This program performs extra checks before computing the signature. In particular, the program additionally checks if the input corresponds to $(\mathsf{acc\text{-}inp}^*, i+1)$. If so, it checks whether $m_{\text{in}} = m_i$ or not, and accordingly outputs either 'A' or 'B' type signature.

Hybrid H_8 In this hybrid, the challenger makes the accumulator 'read enforcing' at position pos^*. The challenger computes $(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, \mathsf{pos}_i)$.

Hybrid H_9 In this hybrid, the challenger outputs an obfuscation of program $W_8 = P_{1,i}\text{-}4\{k'_{\text{sig},A}, k'_{\text{sig},B}, \sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \text{SK}_{D,\text{abo}}, \text{VK}_{D,\text{abo}}, m_i\}$ (defined in Figure 21). This program outputs $\perp$ if on $(i+1)^{th}$ step, the input signature 'A' verifies, and the output state is q_{ac}. Note that the accumulator is 'read enforced' in this hybrid.

Hybrid H_{10} In this hybrid, the challenger uses normal setup for the accumulator related parameters; that is, it computes $(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \text{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc}\ (1^\lambda, T)$. The remaining steps are exactly identical to the corresponding ones in the previous hybrid.

$P_{1,i}$-1

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda$, punctured PRF keys $k'_{\text{sig},A}, k'_{\text{sig},B}$
Signing/Verification keys $\text{SK}_C, \text{VK}_C, \text{SK}_D, \text{VK}_D$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i+1)$, let $r_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$,
   $r_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$,
   $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
   Else set $\text{VK}_A = \text{VK}_C, \text{VK}_B = \text{VK}_D$.
   (b) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (c) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (d) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i)$, let $r'_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, t))$, $r'_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
   Else set $\text{SK}'_A = \text{SK}_C, \text{SK}'_B = \text{SK}_D$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 18: $P_{1,i}$-1

$P_{1,i}$-2

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda$, punctured PRF keys $k'_{\text{sig},A}, k'_{\text{sig},B}$
Signing/Verification keys $\sigma_C, \text{VK}_C, \text{SK}_{\text{abo},D}, \text{VK}_D$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i+1)$, let $r_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1)), r_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$.
   Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
   Else set $\text{VK}_A = \text{VK}_C, \text{VK}_B = \text{VK}_D$.
   (b) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (c) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (d) If $\alpha =$'-' output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i)$, let $r'_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, t)), r'_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \sigma_C$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl\text{-}abo}(\text{SK}'_{\text{abo},D}, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 19: $P_{1,i}$-2

Hybrid H_{11} In this hybrid, the challenger computes $(\sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \sigma_{C,\text{abo}}, \text{VK}_{C,\text{abo}}) = \mathsf{Split}(\text{SK}_C, m_i)$, but does not compute $(\text{SK}_D, \text{VK}_D)$. Instead, it outputs an obfuscation of $W_{10} = P_{1,i}\text{-}4\{k'_{\text{sig},A}, k'_{\text{sig},B}, \sigma_{C,\text{one}},$

$\mathrm{VK}_{C,\mathrm{one}}$, $\mathrm{SK}_{C,\mathrm{abo}}$, $\mathrm{VK}_{C,\mathrm{abo}}$,$m_i$}. Note that the hardwired keys for verification/signing (that is, $\sigma_{C,\mathrm{one}}$, $\mathrm{VK}_{C,\mathrm{one}}$, $\mathrm{SK}_{C,\mathrm{abo}}$,$\mathrm{VK}_{C,\mathrm{abo}}$) are all derived from the same signing key SK_C, whereas in the previous hybrid, the first two components were derived from SK_C while the next two from SK_D.

Hybrid H_{12} In this hybrid, the challenger obfuscates a program $P_{1,i}$-5 (defined in Figure 22) which has a secret key, verification key pair hardwired, instead of the four components in $P_{1,i}$-4. More formally, the challenger chooses $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and outputs an obfuscation of $W_{11} = P_{1,i}\text{-}5\{k'_{\mathrm{sig},A}, k'_{\mathrm{sig},B}, \mathrm{SK}_C, \mathrm{VK}_C\}$.

Hybrid H_{13} In this hybrid, the challenger chooses the randomness r_C used to compute $(\mathrm{SK}_C, \mathrm{VK}_C)$ pseudorandomly; that is, it sets $r_C = \mathsf{F}(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$.

Hybrid H_{14} In this hybrid, the challenger does not guess pos^* at the start of the experiment.

Hybrid H_{13} This corresponds to the hybrid $\mathsf{Hyb}'_{0,j}$-$(1,i)$.

B.3.2 Analysis

We will now show that the consecutive hybrid experiments are computationally indistinguishable. Let $\mathsf{Adv}_y^{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in hybrid experiment H_y.

Claim B.8. For any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_1^{\mathcal{A}} = \mathsf{Adv}_0^{\mathcal{A}}/t^*$.

Proof. This follows from the definition of experiment H_1. The challenger guesses the position read by machine M_{j+1} on input x^* at step i. This guess is correct with probability $1/t^*$, and as a result, if an adversary has advantage ϵ in H_0, then it has advantage ϵ/t^* in H_1. ∎

Claim B.9. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The only difference between H_1 and H_2 is that H_1 uses program $P_{1,i}$, while H_2 uses $P_{1,i}$-1. From the correctness of puncturable PRFs, it follows that both programs have identical functionality for $t \neq i$ or $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$. For $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$, the two programs have identical functionality because $(\mathrm{SK}_C, \mathrm{VK}_C)$ and $(\mathrm{SK}_D, \mathrm{VK}_D)$ are correctly computed using $\mathsf{F}(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$ and $\mathsf{F}(k_{\mathrm{sig},B}, (\mathsf{acc\text{-}inp}^*, i))$ respectively. Therefore, by the security of $i\mathcal{O}$, it follows that the obfuscations of the two programs are computationally indistinguishable. ∎

Claim B.10. Assuming F is a selectively secure puncturable PRF, for any adversary $\mathcal{A}$, $|\mathsf{Adv}_2^{\mathcal{A}} - \mathsf{Adv}_3^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. We will construct an intermediate experiment H, where r_C is chosen uniformly at random, while $r_D = F(k_{\mathrm{sig},B}, (\mathsf{acc\text{-}inp}^*, i))$. Now, if an adversary can distinguish between H_1 and H, then we can construct a reduction algorithm that breaks the security of F. The reduction algorithm sends $(\mathsf{acc\text{-}inp}^*, i)$ as the challenge, and receives $k'_{\mathrm{sig},A}, r$. It then uses r to compute $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r)$. Depending on whether r is truly random or not, $\mathcal{B}$ simulates either hybrid H or H_1. Clearly, if $\mathcal{A}$ can distinguish between H_2 and H with advantage ϵ, then $\mathcal{B}$ breaks the PRF security with advantage ϵ. ∎

Claim B.11. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_3^{\mathcal{A}} - \mathsf{Adv}_4^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This follows from correctness property 2 of splittable signatures. This correctness property ensures that $P_{1,i}$-1 and $P_{1,i}$-2 have identical functionality. ∎

Claim B.12. Assuming $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{one}}$ indistinguishability (Definition 2.3), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_4^{\mathcal{A}} - \mathsf{Adv}_5^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this lemma is similar to the proof of Lemma B.9. Note that the challenger in H_4 and H_5 requires only σ_C. As a result, VK_C and $\mathrm{VK}_{C,\mathrm{one}}$ are computationally indistinguishable. ∎

Claim B.13. Assuming $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{abo}}$ indistinguishability (Definition 2.4), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_5^{\mathcal{A}} - \mathsf{Adv}_6^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This proof is similar to the previous one. If there exists an adversary $\mathcal{A}$ such that $\mathsf{Adv}_5^{\mathcal{A}} - \mathsf{Adv}_6^{\mathcal{A}} = \epsilon$, then there exists a reduction algorithm $\mathcal{B}$ that breaks the $\mathrm{VK}_{\mathrm{abo}}$ security of $\mathcal{S}$ with advantage ϵ. In this case, the reduction algorithm uses the challenger's output to set up $\mathrm{SK}_{D,\mathrm{abo}}$ and VK, which is either VK_D or $\mathrm{VK}_{D,\mathrm{abo}}$. ∎

Claim B.14. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_6^{\mathcal{A}} - \mathsf{Adv}_7^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Let $P_0 = P_{1,i}\text{-}2\{k'_{\mathrm{sig},A}, k'_{\mathrm{sig},B}, \sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{D,\mathrm{abo}}, \mathrm{VK}_{D,\mathrm{abo}}, m_i\}$ and $P_1 = P_{1,i}\text{-}3\{k'_{\mathrm{sig},A}, k'_{\mathrm{sig},B}, \sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{C,\mathrm{abo}}, \mathrm{VK}_{C,\mathrm{abo}}, m_i\}$, where the constants of both programs are computed identically. It suffices to show that P_0 and P_1 have identical functionality. Note that the only inputs where P_0 and P_1 can possibly differ correspond to step $(\mathsf{acc\text{-}inp}^*, i+1)$. Fix any input $(i+1, m_{\mathrm{in}} = (v_{\mathrm{in}}, \mathsf{st}_{\mathrm{in}}, \mathsf{acc}_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}), \mathsf{acc\text{-}inp}, \mathsf{sym}_{\mathrm{in}}, \pi, aux)$. Let us consider two cases:

(a) $m_{\mathrm{in}} = m_i$. In this case, using the correctness properties 1 and 3, we can argue that for both programs, $\alpha =$'A'. Now, P_0 outputs $\mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\mathrm{out}})$, while P_1 is hardwired to output $\mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\mathrm{out}}$. Therefore, both programs have the same output in this case.

(b) $m_{\mathrm{in}} \neq m_i$. Here, we use the correctness property 5 to argue that $\alpha \neq$'A', and correctness properties 2, 1 and 6 to conclude that $\alpha =$'B'. P_1 is hardwired to output $\mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\mathrm{out}})$, while P_0 outputs $\mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\mathrm{out}})$.

∎

Claim B.15. Assuming Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_7^{\mathcal{A}} - \mathsf{Adv}_8^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This follows from Definition 2.6. Suppose, on the contrary, there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}_7^{\mathcal{A}} - \mathsf{Adv}_8^{\mathcal{A}}| = \epsilon$ which is non-negligible in λ. We will construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the Read Setup indistinguishability of Acc. $\mathcal{B}$ first chooses $\mathsf{pos}^* \leftarrow [t^*]$, and sends pos^* to the challenger, and receives $(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0)$. $\mathcal{B}$ uses these components to compute the encoding. Note that the remaining steps are identical in both hybrids, and therefore, $\mathcal{B}$ can simulate them perfectly. Finally, using $\mathcal{A}$'s guess, $\mathcal{B}$ guesses whether the setup was normal or read-enforced. ∎

Claim B.16. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_8^{\mathcal{A}} - \mathsf{Adv}_9^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Let $P_0 = P_{1,i}\text{-}3\{k'_{\mathrm{sig},A}, k'_{\mathrm{sig},B}, \sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{C,\mathrm{abo}}, \mathrm{VK}_{C,\mathrm{abo}}, m_i\}$ and $P_1 = P_{1,i}\text{-}4\{k'_{\mathrm{sig},A}, k'_{\mathrm{sig},B}, \sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{C,\mathrm{abo}}, \mathrm{VK}_{C,\mathrm{abo}}, m_i\}$. We need to show that P_0 and P_1 have identical functionality. Note the only difference could be in the case where $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{C,\mathrm{one}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$ and the remaining inputs are such that $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{ac}}$, then both programs can have different functionality. We will show that this case cannot happen.

From the correctness property 5, it follows that if $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{C,\mathrm{one}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$, then $m_{\mathrm{in}} = m_i$. As a result, $\mathsf{acc}_{\mathrm{in}} = \mathsf{acc}_i$, $\mathsf{pos}_{\mathrm{in}} = \mathsf{pos}_i$, $\mathsf{st}_{\mathrm{in}} = \mathsf{st}_i$. Therefore, $(\mathsf{sym}_{\mathrm{in}} = \epsilon$ **or** $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{sym}_{\mathrm{in}}, \mathsf{acc}_i, \mathsf{pos}_i, \pi) = 1) \implies \mathsf{sym}_{\mathrm{in}} = \mathsf{sym}_i$, which implies $\mathsf{st}_{\mathrm{out}} = \mathsf{st}_{i+1}$. However, since M is not accepting, $\mathsf{st}_{i+1} \neq q_{\mathrm{ac}}$. Therefore, $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{C,\mathrm{one}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$ **and** $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{ac}}$ cannot take place. ∎

Claim B.17. Assuming Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_9^{\mathcal{A}} - \mathsf{Adv}_{10}^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This step is a reversal of the step from H_7 to H_8, and therefore the proof of this claim is similar to that of Claim B.15. ∎

Claim B.18. Assuming $\mathcal{S}$ satisfies splitting indistinguishability (Definition 2.5), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_{10}^{\mathcal{A}} - \mathsf{Adv}_{11}^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. We will use the splittable indistinguishability property (Definition 2.5) for this claim. Assume there is a PPT adversary $\mathcal{A}$ such that $|\mathsf{Adv}_{10}^{\mathcal{A}} - \mathsf{Adv}_{11}^{\mathcal{A}}| = \epsilon$. We will construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the splitting indistinguishability of $\mathcal{S}$. $\mathcal{B}$ first receives as input from the challenger a tuple $(\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}})$, where either all components are derived from the same secret key, or the first two are from one secret key, and the last two from another secret key. Using this tuple, $\mathcal{B}$ can define the constants required for $P_{1,i}$-4. It computes $k'_{\mathrm{sig},A}, k'_{\mathrm{sig},B}$, $\mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}$, m_i as described in hybrid H_{10} and hardwires $\sigma_{\mathrm{one}}, \mathrm{VK}_{\mathrm{one}}, \mathrm{SK}_{\mathrm{abo}}, \mathrm{VK}_{\mathrm{abo}}$ in the program. In this way, $\mathcal{B}$ can simulate either H_{10} or H_{11}, and therefore, use $\mathcal{A}$'s advantage to break the splitting indistinguishability. ∎

Claim B.19. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_{11}^{\mathcal{A}} - \mathsf{Adv}_{12}^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This claim follows from correctness properties of $\mathcal{S}$. Note that the programs W_{10} and W_{11} can possibly differ only if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ or $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$. Let us consider all the possible scenarios. Each of those can be addressed using one/more of the correctness properties of $\mathcal{S}$.

1. Signatures verify and $\mathsf{st}_{\mathrm{in}} = q_{\mathrm{ac}}$. Both programs output $\perp$.
2. $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{C,\mathrm{one}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$ and $\mathsf{st}_{\mathrm{out}} \neq q_{\mathrm{ac}}$. In this case, W_{10} outputs $\mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\mathrm{out}})$. Note that using correctness properties 3 and 5, we get that $m_{\mathrm{in}} = m_i$, and therefore, W_{11} outputs $\mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\mathrm{out}})$.
3. $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{C,\mathrm{one}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 0$ but $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{C,\mathrm{abo}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$. In this case, W_{10} sets $\alpha =$'B', and therefore the program outputs $\mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B,$ $m_{\mathrm{out}})$. Using property 6, it follows that $m_{\mathrm{in}} \neq m_i$, and hence W_{11} also gives the same output.
4. Signatures do not verify at both steps. In this case, both programs output $\perp$.

∎

Claim B.20. Assuming F is a selectively secure puncturable PRF scheme, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_{12}^{\mathcal{A}} - \mathsf{Adv}_{13}^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this claim is identical to the proof of Claim B.10. ∎

Claim B.21. For any adversary $\mathcal{A}$, $\mathsf{Adv}_{13}^{\mathcal{A}} = \mathsf{Adv}_{14}^{\mathcal{A}}/t^*$.

Proof. This proof follows from the definition of the two hybrid experiments. The challenger chooses pos^* at the start of H_{13}, and aborts if this guess is incorrect. In hybrid H_{14}, the challenger does not choose pos^* at the start of the experiment. As a result, if an adversary has advantage ϵ in H_{14}, then it has advantage ϵ/t^* in H_{13}. ∎

Claim B.22. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_{14}^{\mathcal{A}} - \mathsf{Adv}_{15}^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This proof is identical to the proof of Claim B.9; it follows directly from the correctness of puncturable PRFs. ∎

$P_{1,i}$-3

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda$, punctured PRF keys $k'_{\text{sig},A}, k'_{\text{sig},B}$
Signing/Verification keys $\sigma_C, \text{VK}_C, \text{SK}_{\text{abo},D}, \text{VK}_D$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i+1)$, let $r_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1)), r_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$.
   Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
   Else set $\text{VK}_A = \text{VK}_C, \text{VK}_B = \text{VK}_D$.
   (b) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (c) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (d) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i)$, let $r'_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, t)), r'_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \sigma_C$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl\text{-}abo}(\text{SK}'_{\text{abo},D}, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$, $m_{\text{in}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$, $m_{\text{in}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 20: $P_{1,i}$-3

$P_{1,i}$-4

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda$, punctured PRF keys $k'_{\text{sig},A}, k'_{\text{sig},B}$
Signing/Verification keys $\sigma_C, \text{VK}_C, \text{SK}_{\text{abo},D}, \text{VK}_D$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i+1)$, let $r_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1)), r_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$.
   Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
   Else set $\text{VK}_A = \text{VK}_C, \text{VK}_B = \text{VK}_D$.
   (b) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (c) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (d) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i+1$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i)$, let $r'_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, t)), r'_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \sigma_C$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ **and** $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl\text{-}abo}(\text{SK}'_{\text{abo},D}, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 21: $P_{1,i}$-4

$P_{1,i}$-5

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda$, punctured PRF keys $k'_{\text{sig},A}, k'_{\text{sig},B}$
Signing/Verification keys $\underline{\text{SK}_C, \text{VK}_C}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i+1)$, let $r_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1)), r_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$.
   Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
   $\underline{\text{Else set } \text{VK}_A = \text{VK}_C}$.
   (b) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (c) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (d) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i+1$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) If $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i)$, let $r'_A = \mathsf{F}(k'_{\text{sig},A}, (\mathsf{acc\text{-}inp}^*, t)), r'_B = \mathsf{F}(k'_{\text{sig},B}, (\mathsf{acc\text{-}inp}^*, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$, $\underline{\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}_C, m_{\text{out}})}$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 22: $P_{1,i}$-5

B.4 Proof of Lemma B.4

Proof Intuition Let us first note the differences between $P'_{1,i}$ and $P_{1,i+1}$.

Input corr. to	$P'_{1,i}$	$P_{1,i+1}$
$t > t^*$ or $t < i+1$ or $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$	Verify 'A' signatures only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature.	Verify 'A' signatures only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature.
$t = i + 1$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$	Verify 'A' signature only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature if $m_{\text{in}} = m_i$, 'B' signature if $m_{\text{in}} \neq m_i$.	Verify 'A' signature only, output $\perp$ if $\mathsf{st}_{\text{out}} \in \{q_{\text{ac}}, q_{\text{rej}}\}$, else output 'A' signature if $m_{\text{out}} = m_{i+1}$, 'B' signature if $m_{\text{out}} \neq m_{i+1}$.
$i + 2 \leq t \leq t^*$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$	Verify 'A/B' signatures, output $\perp$ if 'B' type signature and $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output signature of same type as incoming signature.	Verify 'A/B' signatures, output $\perp$ if 'B' type signature and $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output signature of same type as incoming signature.

The two programs differ in functionality at $t = i+1$ for inputs corresponding to $\mathsf{acc\text{-}inp}^*$. In program $P'_{1,i}$, the program outputs a good signature only if the input accumulator, iterator and state are the correct ones. We first modify the program to add an additional condition that that the input symbol should also be correct in order to output a good signature. This is enforced using the read-enforcing property of accumulators. Since both the input state and symbol are correct, the output state and symbol are also correct. Next, using the write enforcing property of the accumulator, we can ensure that the signature is good if additionally, the output accumulator is also correct. Finally, by making the iterator enforcing, we can ensure that the program outputs a good signature only if the output accumulator, iterator and state are correct.

Formal Proof We will first define a sequence of hybrid experiments $H_0, \ldots, H_{13}$, where H_0 corresponds to $\mathsf{Hyb}'_{0,j}(1, i)$ and H_{13} corresponds to $\mathsf{Hyb}_{0,j}(1, i+1)$.

Hybrid H_0 This corresponds to $\mathsf{Hyb}'_{0,j}(1, i)$.

Hybrid H_1 This experiment is similar to the previous one, except that the challenger guesses the position read by M_{j+1} on input x^* at step i. More formally, the challenger chooses $\mathsf{pos}^* \leftarrow [t^*]$ at the start of the experiment. On receiving the challenge input x^*, it sends either the PRF evaluation or a truly random string. Next, the adversary sends constrained key queries, and the challenger responds as in H_0. However, on the $(j+1)^{th}$ query M_{j+1}, the challenger aborts if M_{j+1}, on input x^*, does not read position pos^* at step i.

Hybrid H_2 In this hybrid, the challenger uses 'read enforced' setup for the accumulator. The challenger computes $(\text{PP}_{\mathsf{Acc}}, \widetilde{\text{acc}}_0, \widetilde{\text{STORE}_0}) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, \mathsf{pos}^*)$. The remaining steps are same as in the previous hybrid.

Hybrid H_3 In this hybrid, the challenger uses program $W_2 = P'_{1,i}$-1 (defined in Figure 23), which is similar to $P_{1,i}$. However, in addition to checking if $m_{\text{in}} = m_i$, it also checks if $(\mathsf{it}_{\text{out}}, \mathsf{pos}_{\text{out}}, \mathsf{st}_{\text{out}}) = (\mathsf{it}_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1})$.

Hybrid H_4 In this experiment, the challenger uses normal setup instead of 'read enforced' setup for the accumulator.

$P'_{1,i}$-1

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (b) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (c) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i+1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (d) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i+1$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   (b) Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
7. (a) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} = m_i$ **and** $(v_{\text{out}}, \mathsf{pos}_{\text{out}}, \mathsf{st}_{\text{out}}) = (v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1})$,
   $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i{+}1)$ **and** $(m_{\text{in}} \neq m_i$ **or** $(v_{\text{out}}, \mathsf{pos}_{\text{out}}, \mathsf{st}_{\text{out}}) \neq (v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1}))$,
   $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

8. If $t+1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
9. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 23: $P'_{1,i}$-1

Hybrid H_5 This experiment is identical to the previous experiment, except that the challenger guesses the position written at step i. The challenger chooses $\mathsf{pos}^* \leftarrow [t^*]$. On the $(j+1)^{th}$ constrained key query M_{j+1}, the challenger aborts if M_{j+1} on input x^* does not write to position pos^* at step i.

Hybrid H_6 In this hybrid, the challenger 'write enforces' the accumulator at position pos^*. The challenger computes $(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{\mathsf{acc}}_0, \widetilde{\mathrm{STORE}}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}(1^\lambda, T, \mathsf{pos}^*)$. The remaining computation is same as in previous step.

Hybrid H_7 In this experiment, the challenger outputs an obfuscation of $W_5 = P'_{1,i}\text{-}2\{i, k_{\mathrm{sig},A}, k_{\mathrm{sig},B}, m_i, m_{i+1}\}$ (defined in Figure 24), which is similar to W_2. However, on input where $t = i+1$, before computing signature, it also checks if $\mathsf{acc}_{\mathrm{out}} = \mathsf{acc}_{i+1}$. Therefore, it checks whether $m_{\mathrm{in}} = m_i$ and $m_{\mathrm{out}} = m_{i+1}$.

Hybrid H_8 This experiment is similar to the previous one, except that the challenger uses normal setup for accumulator instead of 'enforcing write'.

Hybrid H_9 This experiment is similar to the previous one, except that the challenger uses enforced setup for iterator instead of normal setup. It first computes $\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \mathrm{STORE}_0$ as in the previous hybrid. Next, it computes the first $i+1$ 'correct messages' for the iterator. Let $\mathsf{st}_0 = q_0$ and $\mathsf{pos}_0 = 0$. For $j = 1$ to $i+1$

1. $(\mathsf{sym}_j, \pi_j) = \mathsf{Prep\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{j-1}, \mathsf{pos}_{j-1})$.
2. Let $(\mathsf{st}_j, \mathsf{sym}_{w,j}, \beta) = \delta(\mathsf{st}_{j-1}, \mathsf{sym}_{j-1})$.
3. $aux_j = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{j-1}, \mathsf{pos}_{j-1})$.
4. $\mathsf{acc}_j = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_{j-1}, \mathsf{sym}_{w,j}, \mathsf{pos}_{j-1}, aux_j)$.
5. $\mathrm{STORE}_j = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, \mathrm{STORE}_{j-1}, \mathsf{pos}_{j-1}, \mathsf{sym}_{w,j})$.
6. $\mathsf{pos}_j = \mathsf{pos}_{j-1} + \beta$.

Let $\mathsf{enf} = ((\mathsf{st}_0, w_0, \mathsf{pos}_0), \ldots, (\mathsf{st}_i, w_i, \mathsf{pos}_i))$. It computes $(\mathrm{PP}_{\mathsf{Itr}}, \mathsf{it}_0) \leftarrow \mathsf{Setup\text{-}Itr\text{-}Enforce}(1^\lambda, T, \mathsf{enf})$. The remaining hybrid proceeds as the previous one.

Hybrid H_{10} In this experiment, the challenger outputs an obfuscation of $W_8 = P'_{1,i}\text{-}3\{i, k_{\mathrm{sig},A}, k_{\mathrm{sig},B}, m_{i+1}\}$ (defined in Figure 25), which is similar to W_5, except that it only checks if $m_{\mathrm{out}} = m_{i+1}$.

Hybrid H_{11} The only difference between this experiment and the previous one is that this uses normal Setup for iterator.

Hybrid H_{12} This experiment is identical to the previous one, except that the challenger does not choose pos^* at the start of the experiment, and does not abort depending on the access pattern of M_{j+1} on input x^*.

B.4.1 Analysis

Let $\mathsf{Adv}_i^{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in hybrid H_i.

Claim B.23. For any adversary $\mathcal{A}$, $\mathsf{Adv}_0^{\mathcal{A}} = \mathsf{Adv}_1^{\mathcal{A}}/t^*$.

Proof. The proof of this claim follows from the definition of hybrid H_1. The challenger chooses a uniformly random position $\mathsf{pos}^* \leftarrow [t^*]$ and aborts if M_{j+1}, on input x^*, does not read position pos^* at time i. The challenger's guess is correct with probability $1/t^*$. ∎

Claim B.24. Assuming Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

$P'_{1,i}$-2

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (b) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (c) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i+1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (d) If $\alpha =$'-' output $\perp$.

4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i+1$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} = m_i$ **and** $m_{\text{out}} = m_{i+1}$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $(m_{\text{in}} \neq m_i$ **or** $m_{\text{out}} \neq m_{i+1})$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.

7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 24: $P'_{1,i}$-2

Proof. This proof is identical to the proof of Claim B.15; it follows from Read Setup indistinguishability (Definition 2.6) of Acc. ∎

$P'_{1,i}$-3

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (b) Let $\alpha =$ '-' and $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   (c) If $\alpha =$'-' and ($t > t^*$ **or** $t \leq i+1$ **or** $\mathsf{acc\text{-}inp} \neq \mathsf{acc\text{-}inp}^*$), output $\perp$.
   Else if $\alpha =$'-' and $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   (d) If $\alpha =$'-' output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'B', output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\alpha =$'A' and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq i+1$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$.
   Let $r'_{\text{sig},A} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
   Else if $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$ **and** $m_{\text{in}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}$.

Figure 25: $P'_{1,i}$-3

Claim B.25. Assuming Acc is Read enforcing (Definition 2.8) and $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_2^{\mathcal{A}} - \mathsf{Adv}_3^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. In order to prove this claim, it suffices to show that $P_0 = \mathsf{Prog}'\text{-}2\text{-}i\{i, M, T,$

$\mathsf{msg}_b, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k_{sig,A}, k_{sig,B}, m_i\}$ and $P_1 = \mathsf{Prog}\text{-}2\text{-}i\text{-}b\{i, M, T, \mathsf{msg}_b, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}},$
$k_{sig,A}, k_{sig,B}, m_i, v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1}\}$ are functionally identical. P_0 and P_1 are functionally identical iff $m_{\text{in}} = m_i \implies (v_{\text{out}}, \mathsf{pos}_{\text{out}}, \mathsf{st}_{\text{out}}) = (v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1})$. Here, we will use the Read enforcing property. Note that $m_{\text{in}} = m_i \implies w_{\text{in}} = w_i$, $v_{\text{in}} = v_i$, $\mathsf{st}_{\text{in}} = \mathsf{st}_i$ and $\mathsf{pos}_{\text{in}} = \mathsf{pos}_i$. From Definition 2.8 and the definition of H_1/H_2, it follows that if $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_i, \mathsf{sym}_{\text{in}},$
$\mathsf{pos}_{\text{in}}, \pi) = 1$, then $\mathsf{sym}_{\text{in}} = \mathsf{sym}_i$. This, together with $\mathsf{st}_{\text{in}}, v_{\text{in}}, \mathsf{pos}_{\text{in}}$ implies that $v_{\text{out}} = v_{i+1}$, $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{i+1}$ and $\mathsf{st}_{\text{out}} = \mathsf{st}_{i+1}$. This completes our proof. ∎

Claim B.26. Assuming Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_3^{\mathcal{A}} - \mathsf{Adv}_4^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This step is a reversal of the step from H_1 to H_2; its proof is identical to that of Claim B.9. ∎

Claim B.27. For any adversary $\mathcal{A}$, $\mathsf{Adv}_4^{\mathcal{A}} = \mathsf{Adv}_5^{\mathcal{A}}$.

Proof. Consider an intermediate hybrid experiment where the challenger does not choose pos^*, and therefore does not abort based on the access pattern of M_{j+1} on input x^*. Let $\mathsf{Adv}_{4.5}^{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in this experiment. Then, $\mathsf{Adv}_4^{\mathcal{A}} = \mathsf{Adv}_5^{\mathcal{A}} = \mathsf{Adv}_{4.5}^{\mathcal{A}}$. ∎

Claim B.28. Assuming Acc satisfies indistinguishability of Write Setup (Definition 2.7), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_5^{\mathcal{A}} - \mathsf{Adv}_6^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This proof follows from the Write Setup indistinguishability (Definition 2.7) of Acc. Suppose there exists an adversary $\mathcal{A}$ such that $\mathsf{Adv}_5^{\mathcal{A}} - \mathsf{Adv}_6^{\mathcal{A}} = \epsilon$. We will construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the Write Setup indistinguishability of Acc. $\mathcal{B}$ first chooses $\mathsf{pos}^* \leftarrow [t^*]$ and sends pos^* to the challenger, and receives $\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{\mathrm{STORE}}_0$. The remaining encoding computation is identical in both hybrids, and therefore, $\mathcal{B}$ can simulate it perfectly using $\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{\mathrm{STORE}}_0$. In this manner, $\mathcal{B}$ can perfectly simulate either H_5 or H_6, depending on the challenger's input, and then use $\mathcal{A}$'s response to win the security game with non-negligible advantage. ∎

Claim B.29. Assuming Acc is Write enforcing (Definition 2.9) and $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_6^{\mathcal{A}} - \mathsf{Adv}_7^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Let $P_0 = \mathsf{Prog}\text{-}2\text{-}i\text{-}b\{\mathsf{msg}_b, m_i, v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1}\}$ and $P_1 = \mathsf{Prog}\text{-}2\text{-}i\text{-}c\{\mathsf{msg}_b,\ m_i,\ m_{i+1}\}$. In order to prove that P_0 and P_1 have identical functionality, it suffices to show that $m_{\text{in}} = m_i$ and $(v_{\text{out}}, \mathsf{pos}_{\text{out}}, \mathsf{st}_{\text{out}}) = (v_{i+1},\ \mathsf{pos}_{i+1}, \mathsf{st}_{i+1}) \implies w_{\text{out}} = w_{i+1}$. Here, we will use the Write enforcing property (Definition 2.9). Using the Write enforcing property, we can conclude that $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_i, \mathsf{sym}_{w,i+1}, \mathsf{pos}_i, aux) \implies w_{\text{out}} = w_{i+1}$ or $w_{\text{out}} = \mathit{Reject}$. In either case, we get that the functionality of P_0 and P_1 is identical, therefore implying that their obfuscations are indistinguishable. ∎

Claim B.30. Assuming Acc satisfies indistinguishability of Write Setup (Definition 2.7), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_7^{\mathcal{A}} - \mathsf{Adv}_8^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This step is a reversal of the step from H_4 to H_5; its proof is identical to that of Claim B.28. ∎

Claim B.31. Assuming Itr satisfies indistinguishability of Setup (Definition 2.10), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_8^{\mathcal{A}} - \mathsf{Adv}_9^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}_{\mathcal{A}}^6 - \mathsf{Adv}_{\mathcal{A}}^7|$ is non-negligible. We will construct an algorithm $\mathcal{B}$ that breaks the Setup indistinguishability of Itr (Definition 2.10). $\mathcal{B}$ computes the first $i+1$ tuples to be 'iterated' upon. Let st_j, w_j and pos_j be the state, accumulated value and position after j^{th} step (as described in hybrid H_7). $\mathcal{B}$ sets $\mathsf{enf} = ((\mathsf{st}_0, w_0, \mathsf{pos}_0), \ldots, (\mathsf{st}_i, w_i, \mathsf{pos}_i))$ and sends it to the Itr challenger. It receives $\mathrm{PP}_{\mathsf{Itr}}, v_0$ from the challenger. The remaining computation is identical in both hybrids. Finally, it

sends the encoding to $\mathcal{A}$ and using $\mathcal{A}$'s response, it computes the output to challenger. Since $|\mathsf{Adv}_8^{\mathcal{A}} - \mathsf{Adv}_9^{\mathcal{A}}|$ is non-negligible, $\mathcal{B}$'s advantage is also non-negligible. ∎

Claim B.32. Assuming Itr is enforcing (Definition 2.11) and $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_9^{\mathcal{A}} - \mathsf{Adv}_{10}^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. In order to prove this claim, we need to argue that $P_5 = P_5\{i, k_{sig,A}, k_{sig,B}, m_i, m_{i+1}\}$ and $P_8 = P_8\{i, K_A, K_B, m_{i+1}\}$ are computationally indistinguishable. If we can show that P_5 and P_8 are functionally identical, then using $i\mathcal{O}$ security, we can argue that their obfuscations are computationally indistinguishable. Note that the only difference between P_5 and P_8 is in Step 6: P_5 checks if $(m_{\text{in}} = m_i)$ **and** $(m_{\text{out}} = m_{i+1})$, while P_8 only checks if $(m_{\text{out}} = m_{i+1})$. Therefore, we need to show that $m_{\text{out}} = m_{i+1} \implies m_{\text{in}} = m_i$. This follows directly from the enforcing property of Itr (recall in both hybrids, $\mathrm{PP}_{\mathsf{Itr}}, v_0$ are computed using $\mathsf{Setup\text{-}Itr\text{-}Enforce}$). Since $v_{\text{out}} = v_{i+1}$, it implies $v_{\text{in}} = v_i$ and $(\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}) = (\mathsf{st}_i, w_i, \mathsf{pos}_i)$. This concludes our proof. ∎

Claim B.33. Assuming Itr satisfies indistinguishability of Setup (Definition 2.10), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_{\mathcal{A}}^{10} - \mathsf{Adv}_{\mathcal{A}}^{11}| \leq \mathrm{negl}(\lambda)$.

Proof. This is a reversal of the step from H_8 to H_9, and its proof is similar to that of Claim B.31. ∎

Claim B.34. For any adversary $\mathcal{A}$, $\mathsf{Adv}_{11}^{\mathcal{A}} = \mathsf{Adv}_{12}^{\mathcal{A}}/t^*$.

Proof. This follows directly from the definitions of hybrids H_{11} and H_{12}. The challenger guesses the position written by M_{j+1} on input x^* at step i, and this guess is correct with probability $1/t^*$. ∎

Summing up these claims, it follows that for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_0^{\mathcal{A}} - \mathsf{Adv}_{12}^{\mathcal{A}} \leq t^*(3\epsilon_{i\mathcal{O}} + 4\epsilon_{PA} + 2\epsilon_{\mathsf{Itr}})$.

B.5 Proof of Lemma B.5

Formal Proof We will now describe a sequence of hybrids, where H_0 corresponds to $\mathsf{Hyb}_{0,j}$-$(1, t^* - 1)$ and H_3 corresponds to $\mathsf{Hyb}_{0,j}$-2.

Hybrid H_1 In this hybrid, the challenger guesses the position accessed by machine M_{j+1} on input x^* at step $t^* - 1$. It chooses $\mathsf{pos}^* \leftarrow [t^*]$ at the start of the experiment. Then, it receives the challenge input x^* from the adversary, responds with the PRF evaluation or random string. Next, it receives constrained key queries from the adversary. For the first j queries, it responds as in H_0. On receiving the $(j+1)^{th}$ query, it checks if the guess pos^* is correct. If not, it aborts. Else, it continues the experiment as in H_0.

Hybrid H_2 In this hybrid, the challenger computes the parameters for the accumulator using read-enforced setup at position pos^*. It computes $(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_0, \textsc{store}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, \mathsf{pos}^*)$. The remaining hybrid proceeds as H_1.

Hybrid H_3 In this hybrid, the challenger modfies the $j + 1^{th}$ constrained key. It outputs an obfuscation of $W_2 = P_2\{M_{j+1}, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, k, k_{\text{sig},A}, k_{\text{sig},B}\}$.

Hybrid H_4 In this hybrid, the challenger uses $\mathsf{Setup\text{-}Acc}$ for Acc instead of using $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}$.

Hybrid H_5 In this hybrid. the challenger does not guess pos^* at the start of the experiment, and as a result, does not abort based on the position read by M_{j+1} on input x^* at step $t^* - 1$. Note that this is identical to $\mathsf{Hyb}_{0,j}$-2.

B.5.1 Analysis

Let $\mathsf{Adv}_i^{\mathcal{A}}$ denote the advantage of an adversary $\mathcal{A}$ in hybrid H_i.

Claim B.35. For any adversary $\mathcal{A}$, $\mathsf{Adv}_1^{\mathcal{A}} = \mathsf{Adv}_0^{\mathcal{A}}/t^*$.

Proof. This follows from the definition of H_1. The challenger guesses the position $\mathsf{pos}^* \leftarrow [t^*]$, and this guess matches the position read by M_{j+1} on input x^* at time $t^* - 1$ with probability $1/t^*$. ∎

Claim B.36. Assuming Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. This proof is identical to the proof of Claim B.15; it follows from Read Setup indistinguishability (Definition 2.6) of Acc. ∎

Claim B.37. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_2^{\mathcal{A}} - \mathsf{Adv}_3^{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. To prove this claim, we need to argue that $W_1 = \mathsf{Prog}'\text{-}2\text{-}t^* - 1\{M, T, t^*, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{ltr}}, \mathsf{msg}_b, k_{sig,A}, k_{sig,B}, m_{t^*-1}\}$ and $W_2 = \mathsf{Prog}\text{-}3\{M, T, t^*, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{ltr}}, \mathsf{msg}_b, k_{sig,A}, k_{sig,B}\}$ have identical functionality. The only possible reason for differing functionality is that W_1 could output 'A' type signature when $m_{\mathrm{in}} = m_{t^*-1}$, while W_2 could output 'B' type signature. The critical observation here is that since $m_{\mathrm{in}} = m_{t^*-1}$, both programs output $\perp$. Since $m_{\mathrm{in}} = m_{t^*-1}$, $w_{\mathrm{in}} = w_{t^*-1}$, $\mathsf{pos}_{\mathrm{in}} = \mathsf{pos}_{t^*-1}$ and $\mathsf{st}_{\mathrm{in}} = \mathsf{st}_{t^*-1}$. If we can show that $\mathsf{sym}_{\mathrm{in}} = \mathsf{sym}_{t^*-1}$, then it follows that $\mathsf{st}_{\mathrm{out}} = \mathsf{st}_{t^*} = q_{\mathrm{rej}}$.

Since setup is read enforced at pos_{t^*-1}, there are two possibilities:

1. $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{t^*-1}, \mathsf{sym}_{\mathrm{in}}, \mathsf{pos}_{t^*-1}, \pi) = 0$, in which case both programs output $\perp$.
2. $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{t^*-1}, \mathsf{sym}_{\mathrm{in}}, \mathsf{pos}_{t^*-1}, \pi) = 1$, in which case, $\mathsf{sym}_{\mathrm{in}} = \mathsf{sym}_{t^*-1}$. This implies $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{rej}}$, and therefore, both programs output $\perp$.

Hence, both programs have identical functionality. As a result, by the security of $i\mathcal{O}$, their obfuscations are computationally indistinguishable.

∎

Claim B.38. Assuming Acc satisfies indistinguishability of Read Setup (Definition 2.6), for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_3^{\mathcal{A}} - \mathsf{Adv}_4^{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. This step is a reversal of the step from H_1 to H_2; the proof is identical to the proof of Claim B.15. ∎

Claim B.39. For any adversary $\mathcal{A}$, $\mathsf{Adv}_4^{\mathcal{A}} = \mathsf{Adv}_5^{\mathcal{A}}/t^*$.

Proof. This claim follows from the definition of hybrid H_5. In hybrid H_5, the challenger does not guess the position accessed by M_{j+1} on input x^* at time $t^* - 1$. The adversary's behavior is identical in both the hybrids. ∎

Summing up, for any PPT adversary $\mathcal{A}$, if $\mathsf{Adv}'^{\mathcal{A}}_{1,t^*-1}$ denotes its advantage in $\mathsf{Hybrid}_{0,j}\text{-}(1, t^* - 1)$ and $\mathsf{Adv}_2^{\mathcal{A}}$ its advantage in $\mathsf{Hybrid}_{0,j}\text{-}2$, then $\mathsf{Adv}^{\mathcal{A}}_{1,t^*-1} - \mathsf{Adv}_2^{\mathcal{A}} \leq t^*(\epsilon_{i\mathcal{O}} + 2\epsilon_{PA})$. This concludes the proof of Lemma B.5.

B.6 Proof of Lemma B.6

Recall, t^* represents the running time of machine M_{j+1} on input x^*, and τ^* is the smallest power of two greater than t^*.

Formal Proof We will first define $\tilde{t} = \tau^* - t^* + 1$ intermediate hybrids $H_2\text{-}t^*, \ldots, H_2\text{-}\tau^*$. Then, there are two main parts to the proof. The first one is showing that $H_2\text{-}i$ and $H_2\text{-}(i+1)$ are computationally indistinguishable. This analysis is similar to the tail hybrid analysis in [KLW15], and is presented in Section B.6.1. Next, we need to show that $H_2\text{-}\tau^*$ and Hyb_3 are computationally indistinguishable, which is described in Section B.6.2.

$H_2\text{-}i$ In this hybrid, the challenger outputs an obfuscation of $P_2\text{-}i\{M_{j+1}, k, k_{\mathrm{sig},A}, k_{\mathrm{sig},B}, m_{t^*-1}\}$ (defined in Figure 26).

$P_2\text{-}i$

Constants : Turing machine $M = \langle Q, \Sigma_{\mathrm{tape}}, \delta, q_0, q_{\mathrm{ac}}, q_{\mathrm{rej}} \rangle$, time bound T
Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$,
Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\mathrm{sig},A}, k_{\mathrm{sig},B}$

Inputs : Time t, String seed, position $\mathsf{pos}_{\mathrm{in}}$, symbol $\mathsf{sym}_{\mathrm{in}}$, TM state $\mathsf{st}_{\mathrm{in}}$
Accumulator value $\mathsf{acc}_{\mathrm{in}}$, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value $\mathsf{it}_{\mathrm{in}}$, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\mathrm{PRG}(\mathsf{seed}) \neq \mathrm{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $t^* < t \leq i$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ output $\perp$.
3. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\mathrm{in}}, \mathsf{sym}_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}, \pi) = 0$ output $\perp$.
4. (a) Let $r_{\mathrm{sig},A} = F(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\mathrm{sig},A})$.
   Let $r_{\mathrm{sig},B} = F(k_{\mathrm{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\mathrm{sig},B})$.
   (b) Let $m_{\mathrm{in}} = (\mathsf{it}_{\mathrm{in}}, \mathsf{st}_{\mathrm{in}}, \mathsf{acc}_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 0$ output $\perp$.
5. (a) Let $(\mathsf{st}_{\mathrm{out}}, \mathsf{sym}_{\mathrm{out}}, \beta) = \delta(\mathsf{st}_{\mathrm{in}}, \mathsf{sym}_{\mathrm{in}})$ and $\mathsf{pos}_{\mathrm{out}} = \mathsf{pos}_{\mathrm{in}} + \beta$.
   (b) If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{ac}}$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
6. (a) Compute $\mathsf{acc}_{\mathrm{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\mathrm{in}}, \mathsf{sym}_{\mathrm{out}}, \mathsf{pos}_{\mathrm{in}}, aux)$. If $\mathsf{acc}_{\mathrm{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\mathrm{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, \mathsf{it}_{\mathrm{in}}, (\mathsf{st}_{\mathrm{in}}, \mathsf{acc}_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}))$.
7. (a) Let $r'_{\mathrm{sig},A} = F(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\mathrm{sig},A})$. Let $r'_{\mathrm{sig},B} = F(k_{\mathrm{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\mathrm{sig},B})$.
   (b) Let $m_{\mathrm{out}} = (\mathsf{it}_{\mathrm{out}}, \mathsf{st}_{\mathrm{out}}, \mathsf{acc}_{\mathrm{out}}, \mathsf{pos}_{\mathrm{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, t^*)$, $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\mathrm{out}})$.
   Else $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\mathrm{out}})$.
8. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
9. Output $\mathsf{pos}_{\mathrm{out}}, \mathsf{sym}_{\mathrm{out}}, \mathsf{st}_{\mathrm{out}}, \mathsf{acc}_{\mathrm{out}}, \mathsf{it}_{\mathrm{out}}, \sigma_{\mathrm{out}}, \mathsf{seed}'$.

Figure 26: $P_2\text{-}i$

Note that Programs P_2 and P_2-t^* are functionally identical, and therefore Hyb_2 and H_2-t^* are computationally indistinguishable.

B.6.1 Tail Hybrid - Part 1

In this subsection, we will prove that H_2-i and H_2-$(i+1)$ are computationally indistinguishable for $t^* \leq i < \tau^*$. This step requires the following intermediate hybrid experiments H_2-i-0, ..., H_2-i-5 such that $H_{2,i}$-0 corresponds to H_2-i and H_2-i-5 corresponds to H_2-$(i+1)$.

Hybrid H_2-i-0 : This corresponds to H_2-i.

Hybrid H_2-i-1 : In this hybrid, the challenger first punctures the PRF key $k_{\mathrm{sig},A}$ at input $(\mathsf{acc\text{-}inp}^*, i)$; that is, $k'_{\mathrm{sig},A} \leftarrow \mathsf{F.puncture}(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$. Next, it computes $r_C = F(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$, $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$ and finally, outputs an obfuscation of $P_b = P_2$-i-2$\{k, k'_{\mathrm{sig},A}, k_{\mathrm{sig},B}, \mathrm{VK}_C\}$ (defined in Figure 27). It has verification key VK_C hardwired.

Hybrid H_2-i-2 In this hybrid, the challenger chooses $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ using true randomness instead of pseudorandom string. It then hardwires VK_C in P_2-i-2.

Hybrid H_2-i-3 In this hybrid, the challenger chooses $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ as before, but instead of hardwiring VK_C, it hardwires $\mathrm{VK}_{C,\mathrm{rej}}$ in P_2-i-2; that is, it outputs an obfuscation of P_2-i-2$\{k$, $k'_{\mathrm{sig},A}$, $k_{\mathrm{sig},B}$, m_{t^*-1}, $\mathrm{VK}_{C,\mathrm{rej}}\}$.

Hybrid H_2-i-4 In this hybrid, the challenger chooses $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}})$ using $F(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$. It computes $r_C = F(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$, $(\mathrm{SK}_C, \mathrm{VK}_C,$
$\mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$, $i\mathcal{O}(P_2$-i-2$\{k'_{\mathrm{sig},A}, k_{\mathrm{sig},B}, m_{t^*-1}, \mathrm{VK}_{C,\mathrm{rej}}\})$.

Hybrid H_2-i-5 This hybrid corresponds to H_2-$(i+1)$.

Analysis We will show that H_2-i-y and H_2-i-$(y+1)$ are computationally indistinguishable. Let $\mathsf{Adv}^y_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in H_2-i-y.

Claim B.40. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The only difference between H_2-i-0 and H_2-i-1 is that H_2-i-0 outputs an obfuscation of P_2-i, while H_2-i-1 outputs an obfuscation of P_2-i-2. P_2-i uses puncturable PRF key $k_{\mathrm{sig},A}$, while P_2-i-2 uses punctured key $k'_{\mathrm{sig},A}$ punctured at $(\mathsf{acc\text{-}inp}^*, i)$. P_2-i-2 also has verification key VK_C hardwired, which is computed using $F(k_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$. For $(\mathsf{acc\text{-}inp}, t) \neq (\mathsf{acc\text{-}inp}^*, i+1)$, both programs have identical functionality (this follows from the correctness of puncturable PRFs). For $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$, the verification part is identical, since VK_C hardwired is computed correctly. Also, note that the corresponding secret key is not required at $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i)$ (for input $(\mathsf{acc\text{-}inp}^*, t)$, both programs do not output an 'A' type signature). As a result, the programs have identical functionality. Therefore, this claim follows from the security of $i\mathcal{O}$. ∎

Claim B.41. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this claim follows from the security of puncturable PRFs. ∎

P_2-i-2

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $t^* < t \leq i$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ output $\perp$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
4. (a) If $t \neq i+1$, let $r_A = \mathsf{F.eval}(K_A\{i\}, t-1), r_B = F(K_B, t-1)$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$.
   Else let $\text{VK}_A = \text{VK}$.
   (b) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (c) Let $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
5. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
6. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
7. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$. Let $r'_{\text{sig},A} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, t^*)$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
8. If $t+1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
9. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 27: P_2-i-2

Claim B.42. Assuming $\mathcal{S}$ satisfies VK_{rej} indistinguishability, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. Note that the secret key SK_C is not used in both the hybrids. As a result, if there exists a PPT

adversary $\mathcal{A}$ such that $\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}}$ is non-negligible, then there exists a PPT algorithm that breaks the $\mathrm{VK}_{\mathrm{rej}}$ indistinguishability of $\mathcal{S}$. $\mathcal{B}$ receives a verification key VK from the challenger, which is either a normal verification key or a reject-verification key. It hardwires VK in P_2-i-2. The remaining steps are identical in both hybrids. Based on $\mathcal{A}$'s guess, $\mathcal{B}$ guesses whether VK is a normal verification key or if it always rejects. Since $\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}}$ is non-negligible, $\mathcal{B}$'s advantage is also non-negligible. ∎

Claim B.43. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. This step is the reverse of the step from H_2-i-1 to H_2-i-2; the proof follows from the security of puncturable PRFs. ∎

Claim B.44. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^4_{\mathcal{A}} - \mathsf{Adv}^5_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. The only differences between the programs P_2-i-2 and P_2-$i+1$ are:

1. P_2-i-2 uses a punctured PRF key $k'_{\mathrm{sig},A}$, and has the reject-verification key computed using $\mathsf{F}(k'_{\mathrm{sig},A}, (\mathsf{acc\text{-}inp}^*, i))$. As a result, it outputs $\perp$ for all inputs corresponding to $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$. P_2-$i+1$, on the other hand, uses a puncturable PRF key $k_{\mathrm{sig},A}$, and for inputs corresponding to $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, i+1)$, directly outputs $\perp$.

Using the correctness of puncturable PRFs, and the fact that $\mathrm{VK}_{\mathrm{rej}}$ always outputs $\perp$, we get that the two programs are functionally identical, and therefore, H_2-i-5 and H_2-$(i+1)$ are computationally indistinguishable. ∎

B.6.2 Tail Hybrid: Part 2

In this section, we will show that H_2-τ^* and Hyb_3 are computationally indistinguishable. Let $\tau^* = 2^{\delta^*}$. For this, we will define intermediate hybrids H-y-z for $y \in [\delta^*, \lambda]$ and $z \in [0, 5]$, where H-δ^*-0 corresponds to H_2-τ^*, and H-λ-0 corresponds to Hyb_3.

Sequence of Hybrids

Hybrid H-y-0**:** In this experiment, the challenger outputs an obfuscation of P-y defined in Figure 28.

Hybrid H-y-1 In this hybrid, the key k_y is punctured at input $\mathsf{acc\text{-}inp}^*$. Let $k'_y \leftarrow \mathsf{F.puncture}(k_y, \mathsf{acc\text{-}inp}^*)$ and $\mathsf{chk} = \mathrm{PRG}(\mathsf{F}(k_y, \mathsf{acc\text{-}inp}^*))$. The challenger sends an obfuscation of P-y-1$\{k'_y, \mathsf{chk}\}$ (defined in Figure 29).

Hybrid H-y-2 This hybrid is identical to the previous one, except that in this hybrid, the check string chk is set to be the PRG evaluation at a uniformly random point. More formally, let $\alpha \leftarrow \{0,1\}^\lambda$ and $\mathsf{chk} = \mathrm{PRG}(\alpha)$.

Hybrid H-y-3 This hybrid is identical to the previous one, except that the check string chk is chosen uniformly at random.

Hybrid H-y-4 In this hybrid, the challenger outputs an obfuscation of program P-y-2 defined in Figure 30.

Hybrid H-y-5 In this hybrid, the check string chk is computed as $\mathrm{PRG}(\alpha)$, where α is chosen uniformly at random.

P-y

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.
2. If $t^* < t \leq 2^y$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ output $\perp$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
4. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (b) Let $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
5. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
6. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
7. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$. Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, t^*)$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
8. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \text{␣}$.
9. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 28: *P-y*

Hybrid *H-y-6* In this hybrid, the check string chk is computed as $\text{PRG}(\mathsf{F}(k_y, \mathsf{acc\text{-}inp}^*))$.

Analysis Let $\mathsf{Adv}^{\mathcal{A}}_{y,z}$ denote the advantage of an adversary in hybrid *H-y-z*.

Claim B.45. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{y,0} - \mathsf{Adv}^{\mathcal{A}}_{y,1}| \leq \text{negl}(\lambda)$.

P-y-1

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$
String chk

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. Let δ be an integer such that $2^\delta \leq t < 2^{\delta+1}$.
   If $\delta = y$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $\text{PRG}(\mathsf{seed}) \neq \mathsf{chk}$ output $\perp$.
   Else If $\text{PRG}(\mathsf{seed}) \neq \text{PRG}(\mathsf{F}(k_\delta, \mathsf{acc\text{-}inp}))$ and $t > 1$, output $\perp$.

2. If $t^* < t \leq 2^y$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ output $\perp$.

3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.

4. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (b) Let $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.

5. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.

6. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.

7. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$. Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, t^*)$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.

8. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \sqcup$.
9. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 29: P-y-1

Proof. To prove this claim, it suffices to show that the programs P-y and P-y-1 are computationally indistinguishable. Note the only places k_y is used is in Step 1 (on inputs corresponding to $(\mathsf{acc\text{-}inp}^*, t)$ for $2^y \leq t < 2^{y+1}$) and in Step 8 (on input corresponding to $(\mathsf{acc\text{-}inp}^*, 2^y - 1)$). If an input corresponds to $(\mathsf{acc\text{-}inp}^*, t)$ for $2^y \leq t < 2^{y+1}$, then both programs have identical functionality since chk is computed correctly. For inputs corresponding to $(\mathsf{acc\text{-}inp}^*, 2^y - 1)$, both programs output $\perp$ due to Step 2. This concludes

P-y-2

Constants : Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{ac}}, q_{\text{rej}} \rangle$, time bound T
Public parameters for accumulator PP_{Acc},
Public parameters for Iterator PP_{Itr}
Puncturable PRF keys $k, k_1, \ldots, k_\lambda, k_{\text{sig},A}, k_{\text{sig},B}$
String chk

Inputs : Time t, String seed, position pos_{in}, symbol sym_{in}, TM state st_{in}
Accumulator value acc_{in}, proof π, auxiliary value aux,
accumulation of input $\mathsf{acc\text{-}inp}$, Iterator value it_{in}, signature σ_{in}.

1. If $t^* < t \leq 2^{y+1}$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. (a) Let $r_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},A})$.
   Let $r_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t-1))$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{\text{sig},B})$.
   (b) Let $m_{\text{in}} = (\mathsf{it}_{\text{in}}, \mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
4. (a) Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
   (b) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$ and $\mathsf{acc\text{-}inp} = \mathsf{acc\text{-}inp}^*$ and $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{ac}}$, output $\mathsf{F}(k, \mathsf{acc\text{-}inp})$.
5. (a) Compute $\mathsf{acc}_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, \mathsf{acc}_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $\mathsf{acc}_{\text{out}} = \mathit{Reject}$, output $\perp$.
   (b) Compute $\mathsf{it}_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, \mathsf{it}_{\text{in}}, (\mathsf{st}_{\text{in}}, \mathsf{acc}_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
6. (a) Let $r'_{\text{sig},A} = F(k_{\text{sig},A}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},A})$. Let $r'_{\text{sig},B} = F(k_{\text{sig},B}, (\mathsf{acc\text{-}inp}, t))$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{\text{sig},B})$.
   (b) Let $m_{\text{out}} = (\mathsf{it}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{pos}_{\text{out}})$.
   If $(\mathsf{acc\text{-}inp}, t) = (\mathsf{acc\text{-}inp}^*, t^*)$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
7. If $t + 1 = 2^{\delta'}$, set $\mathsf{seed}' = \mathsf{F}(k_{\delta'}, \mathsf{acc\text{-}inp})$.
   Else, set $\mathsf{seed}' = \lrcorner$.
8. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, \mathsf{acc}_{\text{out}}, \mathsf{it}_{\text{out}}, \sigma_{\text{out}}, \mathsf{seed}'$.

Figure 30: *P-y-2*

our proof. ■

Claim B.46. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{y,1} - \mathsf{Adv}^{\mathcal{A}}_{y,2}| \leq \text{negl}(\lambda)$.

The proof of this claim follows immediately from the security definition of puncturable PRFs.

Claim B.47. Assuming PRG is a secure pseudorandom generator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{y,2} - \mathsf{Adv}^{\mathcal{A}}_{y,3}| \leq \text{negl}(\lambda)$.

Proof. Suppose there exists an adversary such that $\mathsf{Adv}^{\mathcal{A}}_{y,2} - \mathsf{Adv}^{\mathcal{A}}_{y,3} = \epsilon$. Then, we can construct an algorithm $\mathcal{B}$ that breaks the PRG security. $\mathcal{B}$ receives a challenge string chk, where chk is either $\mathrm{PRG}(\alpha)$ for a random string α, or a truly random string. $\mathcal{B}$ can then choose the remaining components required to perfectly simulate either H-y-2 or H-y-3. ∎

Claim B.48. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{y,3} - \mathsf{Adv}^{\mathcal{A}}_{y,4}| \leq \mathrm{negl}(\lambda)$.

Proof. Note that the only inputs on which P-y-1 and P-y-2 can possibly differ correspond to inputs $(\mathsf{acc\text{-}inp}^*, t)$ for $2^y \leq t < 2^{y+1}$. However, for any such input, Step 1 will fail with overwhelming probability because chk is chosen uniformly at random, and therefore, there exists no seed such that $\mathrm{PRG}(\mathsf{seed}) = \mathsf{chk}$. ∎

Claim B.49. Assuming PRG is a secure pseudorandom generator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{y,4} - \mathsf{Adv}^{\mathcal{A}}_{y,5}| \leq \mathrm{negl}(\lambda)$.

This is similar to the proof of Claim B.47.

Claim B.50. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{y,5} - \mathsf{Adv}^{\mathcal{A}}_{y,6}| \leq \mathrm{negl}(\lambda)$.

This also follows directly from the security of puncturable PRFs.

Claim B.51. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{y,6} - \mathsf{Adv}^{\mathcal{A}}_{y+1,0}| \leq \mathrm{negl}(\lambda)$.

Proof. This claim is also immediate from the construction. Note that the only differences are at Step 1 and Step 8. In Step 1, P-y-2 has the correct hardwired check string chk, so the programs are functionally identical there. Also, both programs do not reach Step 8 on inputs corresponding to $(\mathsf{acc\text{-}inp}^*, 2^y - 1)$. Therefore, the programs are functionally identical. ∎