Indistinguishability Obfuscation for Turing Machines with Unbounded Memory

Venkata Koppula
kvenkata@cs.utexas.edu

Allison Bishop Lewko
alewko@cs.columbia.edu

Brent Waters
bwaters@cs.utexas.edu*

Abstract

We show how to build indistinguishability obfuscation ($i\mathcal{O}$) for Turing Machines where the overhead is polynomial in the security parameter λ, machine description $|M|$ and input size $|x|$ (with only a negligible correctness error). In particular, we avoid growing polynomially with the maximum space of a computation. Our construction is based on $i\mathcal{O}$ for circuits, one way functions and injective pseudo random generators.

Our results are based on new "selective enforcement" techniques. Here we first create a primitive called positional accumulators that allows for a small commitment to a much larger storage. The commitment is unconditionally sound for a select piece of the storage. This primitive serves as an "$i\mathcal{O}$-friendly" tool that allows us to make two different programs equivalent at different stages of a proof. The pieces of storage that are selected depend on what hybrid stage we are at in a proof.

We first build up our enforcement ideas in a simpler context of "message hiding encodings" and work our way up to indistinguishability obfuscation.

*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

A code obfuscator takes the description of a program P and compiles it into a functionally equivalent program P' that "hides" the internal logic of P. Recently, there has been a surge of interest in obfuscation with the introduction of the first general purpose obfuscator based on mathematically hard problems by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH$^+$13].

The candidate construction of [GGH$^+$13] allows for obfuscation of any polynomially sized circuit. While circuits represent a general model of computation, they have the drawback that the size of a circuit description of a computation is proportional to the running time of a computation. This might be much longer than the time required to simply describe the computation in a different form. For this reason, we would like to develop and obfuscator for Turing Machines (or other similar models) where the obfuscation time grows polynomially with the machine description size, $|M|$, as opposed to its worst case running time T. In addition to serving as its own application, such a construction would pave the way for other applications such as delegation of computation.

We begin by exploring the possibility of bootstrapping a Turing Machine obfuscator from a circuit obfuscator. Perhaps the most natural approach is to use an obfuscated circuit to perform the step functions of a Turing Machine. Consider the following process for a class of oblivious Turing Machines [PF79] that all share the same tape movements. At each step, the obfuscated program will take in authenticated and encrypted state and tape symbols. It then verifies the signature, decrypts, and calculates the next state and tape symbols which it then encrypts, signs, and outputs. In such a construction, the encoding or obfuscation procedure will be proportional to $|M|$, while the evaluation will call the obfuscated program up to T times. Such an approach can be fairly easily analyzed and proven secure in the Virtual Black Box (VBB) model of obfuscation where we can treat the obfuscated program as an oracle. However, Barak et al. [BGI$^+$12] showed that there exist some functionalities that cannot be VBB obfuscated, thus motivating the current practice of searching for solutions that do not depend on VBB obfuscation.

A initial line of research addressed [BCP14, ABG$^+$13] this problem using a potentially weaker definition of security known as differing-inputs obfuscation ($di\mathcal{O}$). Differing-inputs obfuscation is a "knowledge type" assumption that assumes that if the exists an attack algorithm that distinguishes between obfuscations of two programs P_0, P_1, then there must also exist an efficient extraction algorithm that can find an input x where $P_0(x) \neq P_1(x)$. This functionality is leveraged along with succinct non-interactive arguments of knowledge (SNARKs) [BCCT13] and homomorphic encryption to prove to a short program that a long computation was done correctly, at which point the short program will decrypt. (We remark that this general approach was first explored in the context of witness encryption by Goldwasser et al. [GKP$^+$13]). An advantage of this approach is that it achieves the goal and has no a priori bound on the input size. However, knowledge assumptions are generally considered to be a more risky class of assumptions and Garg, Gentry, Halevi and Wichs [GGHW14] gave recent evidence that there might exist functionalities that cannot be $di\mathcal{O}$ obfuscated. Thus, potentially putting $di\mathcal{O}$ security into a similar situation as VBB. More recently, [IPS14] proposed more restricted form of differing inputs called public-coin differing-inputs obfuscation and showed how a circuit obfuscator could be bootstrapped to a Turing Machine obfuscator with unbounded inputs. An advantage of the public-coin restriction is that it circumvents the implausibility result of [GGHW14]. At the same time, the definition still inherently has a stronger extraction flavor and it is unclear whether security reductions to simpler assumptions under this definition are possible.

Turing Machine Obfuscation from $i\mathcal{O}$

We now turn towards the problem of building TM obfuscation from indistinguishability obfuscation. Recall that if an obfuscator is $i\mathcal{O}$ secure, then the obfuscation of two different programs P_0, P_1 are indistinguishable as long as the programs are functionally equivalent (i.e. $\forall x\ P_0(x) = P_1(x)$). This relatively weaker definition has the advantage that there are no known impossibility (or implausibility) results and there exists progress on proving security under simple assumptions such as multilinear subgroup elimination [GLSW14]. On the other hand, working with "just" $i\mathcal{O}$ presents a new set of challenges since we cannot leverage an oracle interface or an extractor. Instead we must design new tools and techniques that guarantee program functional equivalence at different stages of a proof.

One interesting recent direction is to build an iterated circuit construction where each iteration will take in a machine's previous configuration and output the configuration at the next step. Indeed three recent works [LP14, BGT14, CHJV14] give this approach with the major difference being that [LP14] and [BGT14] use obfuscation of a single program to generate garbled circuits for each iteration and do the garbled evaluation outside of the obfuscated program. In contrast, the first construction (of two) by [CHJV14] does the evaluation inside the obfuscation, which performs an authenticated encryption of the next configuration. For proving security the above constructions roughly work in a hybrid where at proof step i the intermediate computations up to step i are erased and the state at step i is programmed in. The full programming is possible since all of the state is passed on each iteration.

The primary limitation of the above approaches is that the circuit size and thus time both to obfuscate and perform an iteration grows with the maximum configuration size or space of the computation. This is not a problem for computations whose maximum space is close to the size of the machine description or input, but becomes problematic for computations where the space grows significantly larger. Canetti et al. [CHJV14] give a second novel construction where the core obfuscation function only takes in a small memory read at a time, thus an iteration on evaluation is only polynomially dependent on the machine description and the log of the maximum running time T. From a qualitative perspective, the evaluation looks close to mimicking a RAM computation with the step function executed by an obfuscated circuit. However, time to obfuscate the entire RAM program and size of the obfuscated code is still polynomial in the maximum space of the computation. Here the time to initialize the encoding is proportional to the maximum space. For proving security they also use a hybrid which erases the computation's footprint starting from the beginning. However, during the proof instead of programming the i-th state in the i-th slot it can be encoded in the initial setup.

Our Approach

We begin our exploration by realizing a primitive that we call message-hiding encoding. Suppose a party has a TM M, an input x, and a message msg and wishes to give an encoding that will disclose the message to any decoder if $M(x) = 1$. This problem can be considered as a deliberate weakening of garbling or randomized encodings [Yao82, IK00, AIK06] as in Ishai and Wee[IW14] to leak part of the input.[1] Clearly, the encoder could simply evaluate $M(x)$ himself and encode msg if $M(x) = 1$ and $\perp$ otherwise. However, the computation of computing $M(x)$ might be significantly longer than the description of the machine and input. In order to minimize the work of the encoder, we are interested in solutions where the encoding overhead is polynomial in the security parameter λ, machine description $|M|$, maximum input size $|x|$, and $\lg(T)$, where T is a maximum time bound on the computation.[2] In particular, the obfuscating or encoding time and program size only grows polylogarithmically with the maximum space, S, of the computation. Since $T > S$ we do not explicitly include S in our statement.[3] Such a solution gives rise to applications where we want to disclose information based on certain conditions. In addition, the message hiding primitive is sufficient for the application of delegation of computation. To delegate a computation of $M(x)$, the verifier simply needs to perform a message hiding encryption of a random value r of sufficient length. The prover will decode and recover r if the computation accepts, which is sufficient to convince the verifier. To make a scheme publicly verifiable, we can also release $f(r)$ for OWF f on encoding. We note that [LP14, BGT14, CHJV14] make similar observations on delegation of computation. We refer the reader to [KRR14] for further exposition on advances in delegation.

In our approach, we return to the initial idea of having an obfuscated circuit simply perform each TM computation step and sign the input for the next iteration of the computation. In our proof we will proceed

[1] The application of "conditional disclosure" was considered in prior work [GIKM98, AIR01], but solutions were more of a secret sharing type flavor where the conditions were based on which parties combined their shares.

[2] For now and in our constructions we consider a time bound T, but we will shortly discuss ways to remove it.

[3] For simplicity we assume that messages are of length λ. In practice, we could always encode an encryption key K as the message and then use this to tightly encrypt msg.

in a sequence of hybrids where in Hybrid i the program is "hardwired" to demand at timestep $t = i$ that the output m_{out} match the honestly computed value m_i^*. The goal is to iterate the proof hardwiring until the step t^* where the honest computation will hit the reject state and halt. As we move along from Hybrid i to Hybrid $i+1$, it is important that the proof will "clean up" the hardwiring on step i as it places new hardwiring on step $i+1$. Otherwise, the final circuit of the proof would contain hardwirings for all t^* steps of the computation. If this occured, the obfuscated step circuit of the real computation would need to be padded to this length to make $i\mathcal{O}$ arguments go through, which would defeat the entire point of aiming for Turing Machines.

A central difficulty comes from the fact that each computational step i will not only require the signed TM state computed at step $i-1$, but also needs a tape symbol that may have been written at a much earlier time period. The main challenge is how to enforce that the hardwiring of the output of step $i-1$ can be translated into a correct wiring of step i without having the obfuscated program pass the entire storage (i.e. tape configuration) at each step. In addition, we wish to avoid any encoding which requires programming proportional to the maximum storage.

To overcome this challenge, we introduce our technique of "selective enforcement". Here we first create a primitive called a positional accumulator that allows for a small commitment to a much larger storage. The commitment is unconditionally sound for a select piece of the storage. Moreover, we are able to computationally hide which portion of the storage is unconditionally enforced. The idea of the selective enforcement primitive is to use such an "$i\mathcal{O}$-friendly" tool that allows us to make two different programs equivalent at different stages of a proof. In particular, if we are at hybrid step i which reads tape position pos, then we want unconditional soundness at this tape position.

More specifically, a positional accumulator behaves somewhat similarly to standard notations of accumulators [BdM93]. It has a setup algorithm $\mathsf{Setup\text{-}Acc}(1^\lambda, S)$ that takes in a security parameter (in unary) and a maximum storage size S in binary. It outputs parameters PP, an initial accumulator value w_0, and storage value STORE_0. In addition, the system has algorithms for writing to the storage, updating the accumulator to reflect writes, and proving and verifying reads. Writes will be of a message m to an index in $[0, S-1]$ and reads will be from a certain index. The accumulator value will stay small while the storage will grow with the number of unique indices written to. Similarly, the update and verify read algorithms should be polynomial in $\lg(S)$ and λ. In addition to the normal setup, there is a separate setup algorithm $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, (m_1, \text{INDEX}_1), \ldots, (m_k, \text{INDEX}_k), \text{INDEX}^*)$. This algorithm takes in a sequence of message and index pairs as well as a special INDEX^*. Informally, this setup has the property that if one writes the messages in the order given (and updates the accumulator from w_0 to w_k honestly) then it is unconditionally impossible to give a false proof for $*$ on accumulator value w_k. Moreover, the output of this setup algorithm is computationally indistinguishable from the output of a standard setup. There is also an analogous setup algorithm for selectively enforcing correct updates on the accumulator. Our construction uses $i\mathcal{O}$, puncturable PRFs, and public key encryption — thus it can be done from $i\mathcal{O}$ and one way functions. The construction uses a form of a Merkle hash tree that fills out the Merkle tree dynamically, using an obfuscated circuit as the main hash function. We defer further details to Section 4.

With these ideas in place, we can describe our main proof steps at a high level. At Hybrid i the obfuscated step circuit will be hardwired to only output a "good" signature if the derived output is the correct tuple $(\mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}\mathsf{pos}_{\text{out}})$ representing the correct output TM state, positional accumulator value, iterator value, and tape position.[4]

We next transition to an intermediate hybrid where $(\mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}\mathsf{pos}_{\text{out}})$ are hardwired into the *input* of the next stage $i+1$. Thus the changes from this proof step move across iterations. Executing this step involves many small hybrids and the use of another $i\mathcal{O}$ friendly tool that we introduce called splittable signatures. We again defer further details to the main body, but we emphasize that we execute this transition without utilizing complexity leveraging and loose only polynomial factors in the reduction.

To complete the hybrid argument we need to be able to transition from hardwiring the inputs of step $i+1$ to hardwiring the outputs. This is where the selective enforcement enters the picture. In general,

[4]We also define a second tool of an iterator that is allows for a different kind of selective enforcement. We defer further explanation to the main body.

using a normal setup does not guarantee the equivalence of hardwiring the correct inputs versus hardwiring the correct outputs. However, if we first (indistinguishably) change the accumulator and iterator to be unconditionally enforcing at the right places, then we can use $i\mathcal{O}$ to change the hardwiring from the input to the output. We then cleanup by changing the accumulator and iterator to normal setup. In contrast to the last, this proof step makes its hardwiring changes within a timestep.

Taken all together, our proof can iterate through these hybrids until we can change the hardwiring to the step t^* and then use $i\mathcal{O}$ to erase the message from the program. We make a few remarks. The first is that in message hiding encoding, the decoder can learn the machine's computation and state — only the message needs to be hidden. Therefore, our computation can follow the real one rather closely and there is no need to absorb the additional overhead of making the computation oblivious. Second, for concreteness we chose our construction to be in the Turing Machine model, however, we believe it could be fairly easily adjusted to a different model such as RAM by simply letting the next position be a function of the state into [0, S-1] as opposed to moving the head position. In particular, our proof structure would remain the same with this minor change to the construction.

Finally, we consider the time bound T. In practice we could set T to be 2^λ. Since the overhead of obfuscating is a fixed polynomial in $\lg(T)$ and λ, applying this setting would give a fixed polynomial in λ. At the same time the construction would then work for computations whose running time was any polynomial in λ. Note that this may result in a negligible correctness error. Consider a class of computations whose running time is some polynomial function $p(\cdot)$. There must exists some constant λ_0 such that for all $\lambda \geq \lambda_0$ we have $p(\lambda) \leq 2^\lambda$ and the system will be correct for all those values. Thus a particular poly-time computation will only incorrect for a constant number of λ values and have neglgible error asymptotically.

We can also remove the time bound altogether if there exists an encryption scheme that is secure against all attackers running in time polynomial in λ, but where the ciphertexts can be decrypted by an algorithm running in time polynomial in 2^λ. Systems such as Elliptic-Curve ElGamal are believed to have this property. The idea is to create a side encryption of msg under the encryption system. During decoding the computation will run the regular system for up to $T = 2^\lambda$ steps, if the computation still has not halted it will switch to brute force decrypting the side ciphertext which will take polynomial in T time.

Hiding the Computation

We now move toward the more complex case of hiding the computation. Here we consider a primitive we call machine hiding encoding. We consider Turing Machines whose tape movements can be calculated by a function $d(t)$ of the time step. A party will have M and an input x and wishes to encode $M(x)$. The security property we desire is that if there exist two machines M_0 and M_1 that share the same tape movement function $d()$ and $M_0(x) = M_1(x)$, and they halt in the same number of steps t^*, then it is difficult to tell apart an encoding of (M_0, x) from (M_1, x). We note this problem can be viewed as indistinguishability obfuscation of Turing Machines restricted to one input. In addition, it can readily be realized by a randomized encoding scheme (with similar efficiency properties) using a Universal Turing Machine. We found this formulation easiest to work with for our construction.

Our new construction follows the previous one closely, but with two important differences. First, the tape movement is a function $d()$ of the time step. By translating to Oblivious Turing Machines [PF79] we can make all head movements the same. Second, instead of writing the state and tape symbols in the clear they will be encrypted by the obfuscated step circuit under a public key derived (via a puncturable PRF) from the time step t on which they were written.

To prove security we will show that an encoding of (M_0, x) and also (M_1, x) are both indistinguishable from a simulated encoding that does not reflect either computation other than the common output, input, tape movement, and time taken. Therefore both encodings will be indistinguishable from each other.

We will prove this by a hybrid which erases both the machine logic inside the obfuscated program as well as the state and tape symbols it writes out. The high level strategy is to iteratively replace the normal logic at step i with a program that on any valid signature will simply output an encryption of a distinct erase symbol erase to both the state and tape. We first observe that such a hybrid strategy will not work well if it follows the computation in the forward direction from start to finish. Suppose we have just erased the logic of the i-th step, then the i-th step will write erase to a tape position which is read at some later time step $j > i$. However, since time step j has not yet been changed, it will not know how to handle the erase symbol and the proof cannot proceed.

For this reason, our proof proceeds in the reverse direction from Hybrid t^* to Hybrid 0, where t^* is the number of steps both computations take. At Hybrid i all steps j where $i \leq j < t^*$ are hardwired to be erasing. Step t^* is wired to the output $M_0(x)$ and steps $j > t^*$ simply abort. Steps $j < i$ are as in the real construction.

The proof proceeds with two major steps. First we define an intermediate variant of Hybrid i where steps $j \geq i$ are erasing as describe above and steps $j < i - 1$ are as in the normal scheme. The significant change is that step $i - 1$ is hardwired to the correct output of the computation. We do this by essentially using our proof techniques from the message-hiding case as a subroutine for hardwiring the correct output at $i - 1$. Next, we use the security of the encryption scheme plus some $i\mathcal{O}$ tricks to switch $i - 1$ from the correct computation to an erasing one, thus moving down to Hybrid $i - 1$. Once we get to the bottom hybrid step, the proof is complete. The main theme is that the enforcement ideas from before plus a little additional encryption is enough to hide the computation.

Going to Indistinguishability Obfuscation

We finally complete things by sketching how machine hiding encoding implies indistinguishability obfuscation for Turing Machines. To do so we use the notion of positional indistinguishability obfuscation [GLSW14, GLW14] adapted to Turing Machines. The process is fairly straightforward and similar transformation are seen in [LP14, BGT14, CHJV14]. Here an obfuscator takes in two Turing Machine descriptions M_0, M_1 (of bounded input size) with a common tape movement function $d()$ and an index $j \in [0, 2^x]$ to produce an obfuscated program P. The program $P(x)$ should output $M_0(x)$ for all inputs $x \geq j$ and output $M_1(x)$ for all inputs $x < j$. Such a scheme will be positionally secure if for all inputs j where $M_0(j) = M_1(j)$ that take the same number of time steps to halt, it should be hard to distinguish between an obfuscation to index j and $j + 1$. It can be shown by a simple hybrid argument that positional indistinguishability obfuscation implies standard $i\mathcal{O}$ with a $2^{|x|}$ loss in the hybrid. To compensate for the exponential loss, we must use complexity leveraging and an underlying positional $i\mathcal{O}$ scheme with subexponential hardness.

We observe that our machine hiding encoding and $i\mathcal{O}$ for circuits give immediate rise to positional $i\mathcal{O}$. Simply obfuscate the following program: on input x create a machine hiding encoding for (M_b, x) where the bit $b = 0$ iff $b \geq j$, where j is the index for obfuscation. The encoding is done with randomness derived from a puncturable PRF.

1.1 Organization

In Section 2 we give preliminaries. In Sections 3, 4 and 5 we give the definitions, constructions and proofs of the respective building blocks of iterators, positional accumulators, and splittable signatures. In Section 6 we give our message hiding encoding construction and proof. In Section 7 we show how to do machine hiding encoding.

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 $M = \langle Q, \Sigma_{\text{tape}}, \Sigma_{\text{inp}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$ where Q and Σ_{tape} are finite sets with the following properties:

- Q is the set of finite states.

- Σ_{inp} is the set of input 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 '$\llcorner\!\lrcorner$' $\in \Sigma_{\text{tape}} \setminus \Sigma_{\text{inp}}$.

- $\delta : Q \times \Sigma_{\text{tape}} \to Q \times \Sigma_{\text{tape}} \times \{+1, -1\}$ is the transition function.

- $q_0 \in Q$ is the start state.

- $q_{\text{acc}} \in Q$ is the accept state.

- $q_{\text{rej}} \in Q$ is the reject state, where $q_{\text{acc}} \neq q_{\text{rej}}$.

For any $i \leq T$, we define the following variables:

- $M^T_{\text{tape},i} \in \Sigma^T_{\text{tape}}$: A T dimensional vector which gives the description of tape before i^{th} step.

- $M^T_{\text{pos},i}$: An integer which describes the position of Turing machine header before i^{th} step.

- $M^T_{\text{state},i} \in Q$: This denotes the state of the Turing machine before step i.

Initially, $M^T_{\text{tape},1} = (\llcorner\!\lrcorner)^T$, $M^T_{\text{pos},1} = 0$ and $M^T_{\text{state},1} = q_0$. At each time step, the Turing machine reads the tape at the head position and based on the current state, computes what needs to be written on the tape, the next state and whether the header must move left or right. More formally, let $(q, \alpha, \beta) = \delta(M^T_{\text{state},i}, M^T_{\text{tape},i}[M^T_{\text{pos},i}])$. Then, $M^T_{\text{tape},i+1}[M^T_{\text{pos},i}] = \alpha$, $M^T_{\text{pos},i+1} = M^T_{\text{pos},i} + \beta$ and $M^T_{\text{state},i+1} = q$. M accepts at time t if $M^T_{\text{state},t+1} = q_{\text{acc}}$. Given any Turing machine M and time bound T, let $\Pi^T_M = 1$ if M accepts within T steps, else $\Pi^T_M = 0$.

2.2 Puncturable Pseudorandom Functions

The notion of constrained PRFs was introduced in the concurrent works of [BW13, BGI14, KPTZ13]. Punctured PRFs, first termed by [SW14] are a special class of constrained PRFs.

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

- $F.\mathsf{setup}(1^\lambda)$ 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.

- $F.\mathsf{puncture}(K, x)$ is a randomized algorithm that takes as input a PRF key $K \in \mathcal{K}$ and $x \in \mathcal{X}$, and outputs a key $K_x \in \mathcal{K}_p$.

- $F.\mathsf{eval}(K_x, x')$ 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 $K_x \leftarrow F.\mathsf{puncture}(K, x)$. For correctness, we need the following property:

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

In this work, we will only need selectively secure puncturable PRFs. The selective security game between the challenger and the adversary A consists of the following phases.

Challenge Phase A sends a challenge $x^* \in \mathcal{X}$. The challenger chooses uniformly at random a PRF key $K \leftarrow \mathcal{K}$ and a bit $b \leftarrow \{0, 1\}$. It computes $K\{x^*\} \leftarrow F.\mathsf{puncture}(K, x^*)$. If $b = 0$, the challenger sets $y = F(K, x^*)$, else $y \leftarrow \mathcal{Y}$. It sends $K\{x^*\}, y$ to A.

Guess A outputs a guess b' of b.

A wins if $b = b'$. The advantage of A is defined to be $\mathsf{Adv}^F_A(\lambda) = \Pr[A \text{ wins}]$.

Definition 2.1. The PRF F is a selectively secure puncturable PRF if for all probabilistic polynomial time adversaries $\mathcal{A}$, $\mathsf{Adv}^F_{\mathcal{A}}(\lambda)$ is negligible in λ.

2.3 Obfuscation

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

Definition 2.2. (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 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

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

In a recent work, [GGH$^+$13] 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.

3 Iterators

In this section, we define the notion of *cryptographic iterators* and show a construction based on indistinguishability obfuscators, selectively secure puncturable PRFs, and $\mathsf{IND\text{-}CPA}$ secure $\mathcal{PKE}$. 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 $v_0 \in \mathcal{S}$.

$\mathsf{Iterate}(\mathrm{PP}, v_{\mathsf{in}}, m)$ 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 $k \le T$, we will use the notation $\mathsf{Iterate}^k(\mathrm{PP}, v_0, (m_1, \ldots, m_k))$ to denote $\mathsf{Iterate}(\mathrm{PP}, v_{k-1}, m_k)$, where $v_j = \mathsf{Iterate}(\mathrm{PP}, v_{j-1}, m_j)$ for all $1 \le j \le k-1$.

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 3.1 (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.

Exp-Setup-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 $(\mathrm{PP}, v_0) \leftarrow \mathsf{Setup\text{-}Itr}(1^\lambda, T)$. Else, it outputs $(\mathrm{PP}, v_0) \leftarrow \mathsf{Setup\text{-}Itr\text{-}Enforce}(1^\lambda, T, 1^k, \mathbf{m} = (m_1, \ldots, m_k))$.
4. $\mathcal{A}$ sends a bit b'.

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

Definition 3.2 (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 $1 \le j \le k$. 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.

3.1 Construction

We will now describe our iterator construction. The main idea is that states (generated by the normal setup algorithm) will correspond to ciphertexts encrypting 0 in a $\mathcal{PKE}$ scheme, and the public parameters will be an obfuscated program that computes randomness for a new encryption by evaluating a puncturable PRF on the current state. To enforce the information-theoretic property at a particular state, we will use a sequence of hybrids to switch to using a punctured key, replace the randomness at that punctured point with a fresh value, and exchange a hardwired fresh encryption of 0 with a fresh encryption of 1. Perfect correctness for the $\mathcal{PKE}$ scheme will ensure that this value cannot be obtained at any other input, since all other inputs will produce encryptions of 0.

Let $i\mathcal{O}$ be an indistinguishability obfuscator, $\mathcal{PKE} = (\mathsf{PKE.setup}, \mathsf{PKE.enc}, \mathsf{PKE.dec})$ a public key encryption scheme with message space $\{0,1\}$ and ciphertext space $\mathcal{C}_{\mathsf{PKE}}$. We will assume $\mathsf{PKE.enc}$ uses r bits of randomness. Let F a puncturable pseudorandom function with key space $\mathcal{K}$, punctured key space $\mathcal{K}_p$, domain $\{0,1\}^\lambda$, range $\{0,1\}^r$ and algorithms $F.\mathsf{setup}, F.\mathsf{eval}, F.\mathsf{puncture}$. Our iterator $\mathcal{I} = (\mathsf{Setup\text{-}Itr}, \mathsf{Setup\text{-}Itr\text{-}Enforce}, \mathsf{Iterate})$ with message space $\{0,1\}^\ell$ and state space $\mathcal{C}_{\mathsf{PKE}} \times \mathbb{N}$ is defined as follows.

- $\mathsf{Setup\text{-}Itr}(1^\lambda, T)$: The setup algorithm chooses $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{PKE.setup}(1^\lambda)$ and puncturable PRF key $K \leftarrow F.\mathsf{setup}(1^\lambda)$. It sets $\mathrm{PP} \leftarrow i\mathcal{O}(\mathsf{Prog}\{K, \mathrm{PK}\})$, where Prog is defined in Figure 1. Let $\mathsf{ct}_0 \leftarrow \mathsf{PKE.enc}(\mathsf{pk}, 0)$. The initial state $v_0 = (\mathsf{ct}_0, 0)$.

Program Prog

Constants: Puncturable PRF key K, $\mathcal{PKE}$ public key PK.
Input: State $v_{\mathsf{in}} = (\mathsf{ct}_{\text{in}}, j) \in \mathcal{C}_{\mathsf{PKE}} \times [T]$, message $m \in \{0,1\}^{\ell}$.

1. Compute $r = F(K, (v_{\mathsf{in}}, m))$.
2. Let $\mathsf{ct}_{\text{out}} = \mathsf{PKE.enc}(\text{PK}, 0; r)$. Output $v_{\mathsf{out}} = (\mathsf{ct}_{\text{out}}, j+1)$.

Figure 1: Program Prog

- $\mathsf{Setup\text{-}Itr\text{-}Enforce}(1^{\lambda}, T, 1^k, \mathbf{m} = (m_1, \ldots, m_k))$: The 'enforced' setup algorithm chooses $(\text{PK}, \text{SK}) \leftarrow \mathsf{PKE.setup}(1^{\lambda})$. Next, it chooses $K \leftarrow F.\mathsf{setup}(1^{\lambda})$. It computes $\mathsf{ct}_0 \leftarrow \mathsf{PKE.enc}(\text{PK}, 0)$ and sets $v_0 = (\mathsf{ct}_0, 0)$. For $j = 1$ to $k-1$, it computes $r_j = F(K, (v_{j-1}, m_j))$, $\mathsf{ct}_j = \mathsf{PKE.enc}(\text{PK}, 0; r_j)$ and sets $v_j = (\mathsf{ct}_j, j)$. It computes a punctured key $K\{(v_{k-1}, m_k)\} \leftarrow F.\mathsf{puncture}(K, (v_{k-1}, m_k))$, chooses $r_k \leftarrow \{0,1\}^r$ and sets $\mathsf{ct}_k = \mathsf{PKE.enc}(\text{PK}, 1; r_k)$. Finally, it computes the public parameters $\text{PP} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}Enforce}\{k, v_{k-1}, m_k, K\{(v_{k-1}, m_k)\}, \mathsf{ct}_k, \text{PK}\})$, where $\mathsf{Prog\text{-}Enforce}$ is defined in Figure 2.

Program $\mathsf{Prog\text{-}Enforce}$

Constants: Integer $k \in [T]$, state v_{k-1}, message $m_k \in \{0,1\}^{\ell}$, Puncturable PRF key $K\{(v_{k-1}, m_k)\}$, $\mathsf{ct}_k \in \mathcal{C}_{\mathsf{PKE}}$, $\mathcal{PKE}$ public key PK.
Input: State $v_{\mathsf{in}} = (\mathsf{ct}_{\text{in}}, j) \in \mathcal{C}_{\mathsf{PKE}} \times [T]$, message $m \in \{0,1\}^{\ell}$.

1. If $v_{\mathsf{in}} = v_{k-1}$ and $m = m_k$, output $v_{\mathsf{out}} = (\mathsf{ct}_k, k)$
2. Else, compute $r = F(K, (v_{\mathsf{in}}, m))$.
3. Let $\mathsf{ct}_{\text{out}} = \mathsf{PKE.enc}(\text{PK}, 0; r)$. Output $v_{\mathsf{out}} = (\mathsf{ct}_{\text{out}}, j+1)$.

Figure 2: Program $\mathsf{Prog\text{-}Enforce}$

- $\mathsf{Iterate}(\text{PP}, v_{\mathsf{in}}, m)$: The iterator algorithm simply outputs $\text{PP}(v_{\mathsf{in}}, m)$.

Efficiency The construction runs in time polynomial of $\lg(T)$ and λ.

3.2 Security

We will now show that the construction described in Section 3.1 satisfies indistinguishability of Setup phase and is enforcing.

Lemma 3.1 (Indistinguishability of Setup Phase)**.** Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, $\mathcal{PKE}$ is a $\mathsf{IND\text{-}CPA}$ secure public key encryption scheme and F is a selectively secure puncturable pseudorandom function, any PPT adversary $\mathcal{A}$ has at most negligible advantage in the $\mathsf{Exp\text{-}Setup\text{-}Itr}$ security game.

Proof. In order to prove this lemma, we will define a sequence of intermediate hybrid experiments $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_3$. Hyb_0 corresponds to the case where the challenger sends public parameters PP generated using $\mathsf{Setup\text{-}Itr}$, while Hyb_3 corresponds to the case where the challenger sends PP generated using $\mathsf{Setup\text{-}Itr\text{-}Enforce}$.

Hyb_0 In this experiment, the challenger computes PP using $\mathsf{Setup\text{-}Itr}$.

1. $\mathcal{A}$ sends $T \in \Theta(2^{\lambda})$.
2. $\mathcal{A}$ sends k messages $m_1, \ldots, m_k \in \{0,1\}^{\ell}$.
3. Challenger chooses $(\text{PK}, \text{SK}) \leftarrow \mathsf{PKE.setup}(1^{\lambda})$ and $K \leftarrow F.\mathsf{setup}(1^{\lambda})$.
   It chooses $r_0 \leftarrow \{0,1\}^r$, sets $\mathsf{ct}_0 = \mathsf{PKE.enc}(\text{PK}, 0; r)$ and $v_0 = (\mathsf{ct}_0, 0)$.
   It sets $\text{PP} \leftarrow i\mathcal{O}(\mathsf{Prog}\{K, \text{PK}\})$ and sends (PP, v_0) to $\mathcal{A}$.
4. $\mathcal{A}$ sends a bit b'.

Hyb_1 This experiment is similar to the previous one, except that the challenger sends an obfuscation of Prog-Enforce. The program Prog-Enforce uses a punctured PRF key, and has the PRF output hardwired at the punctured point.

1. $\mathcal{A}$ sends $T \in \Theta(2^\lambda)$.
2. $\mathcal{A}$ sends k messages $m_1, \ldots, m_k \in \{0,1\}^\ell$.
3. Challenger chooses $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{PKE.setup}(1^\lambda)$ and $K \leftarrow F.\mathsf{setup}(1^\lambda)$.
   It chooses $r_0 \leftarrow \{0,1\}^r$, sets $\mathsf{ct}_0 = \mathsf{PKE.enc}(\mathrm{PK}, 0; r_0)$ and $v_0 = (\mathsf{ct}_0, 0)$.
   Next, it computes $r_j = F(K, (v_{j-1}, m_j))$, sets $\mathsf{ct}_j = \mathsf{PKE.enc}(\mathrm{PK}, 0; r_j)$ and $v_j = (\mathsf{ct}_j, j)$ for all $j \leq k$.
   It computes a punctured PRF key $K\{(v_{k-1}, m_{k-1})\} \leftarrow F.\mathsf{puncture}(K, (v_{k-1}, m_{k-1}))$.
   It sets $\mathrm{PP} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}Enforce}\{k, v_{k-1}, m_{k-1}, K\{(v_{k-1}, m_{k-1})\}, \mathsf{ct}_k, \mathrm{PK}\})$ and sends (PP, v_0) to $\mathcal{A}$.
4. $\mathcal{A}$ sends a bit b'.

Hyb_2 In this hybrid experiment, the ciphertext ct_k is computed using true randomness, instead of a pseudorandom string.

1. $\mathcal{A}$ sends $T \in \Theta(2^\lambda)$.
2. $\mathcal{A}$ sends k messages $m_1, \ldots, m_k \in \{0,1\}^\ell$.
3. Challenger chooses $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{PKE.setup}(1^\lambda)$ and $K \leftarrow F.\mathsf{setup}(1^\lambda)$.
   It chooses $r_0 \leftarrow \{0,1\}^r$, sets $\mathsf{ct}_0 = \mathsf{PKE.enc}(\mathrm{PK}, 0; r_0)$ and $v_0 = (\mathsf{ct}_0, 0)$.
   Next, it computes $r_j = F(K, (v_{j-1}, m_j))$, sets $\mathsf{ct}_j = \mathsf{PKE.enc}(\mathrm{PK}, 0; r_j)$ and $v_j = (\mathsf{ct}_j, j)$ for all $j < k$.
   For $j = k$, it chooses $r_k \leftarrow \{0,1\}^r$, sets $\mathsf{ct}_k = \mathsf{PKE.enc}(\mathrm{PK}, 0; r_k)$ and $v_k = (\mathsf{ct}_k, k)$.
   It computes a punctured PRF key $K\{(v_{k-1}, m_{k-1})\} \leftarrow F.\mathsf{puncture}(K, (v_{k-1}, m_{k-1}))$.
   It sets $\mathrm{PP} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}Enforce}\{k, v_{k-1}, m_{k-1}, K\{(v_{k-1}, m_{k-1})\}, \mathsf{ct}_k, \mathrm{PK}\})$ and sends (PP, v_0) to $\mathcal{A}$.
4. $\mathcal{A}$ sends a bit b'.

Hyb_3 In this experiment, the challenger outputs PP computed using Setup-Itr-Enforce. It is similar to the previous experiment, except that ct_k is an encryption of 1.

1. $\mathcal{A}$ sends $T \in \Theta(2^\lambda)$.
2. $\mathcal{A}$ sends k messages $m_1, \ldots, m_k \in \{0,1\}^\ell$.
3. Challenger chooses $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{PKE.setup}(1^\lambda)$ and $K \leftarrow F.\mathsf{setup}(1^\lambda)$.
   It chooses $r_0 \leftarrow \{0,1\}^r$, sets $\mathsf{ct}_0 = \mathsf{PKE.enc}(\mathrm{PK}, 0; r_0)$ and $v_0 = (\mathsf{ct}_0, 0)$.
   Next, it computes $r_j = F(K, (v_{j-1}, m_j))$, sets $\mathsf{ct}_j = \mathsf{PKE.enc}(\mathrm{PK}, 0; r_j)$ and $v_j = (\mathsf{ct}_j, j)$ for all $j < k$.
   For $j = k$, it chooses $r_k \leftarrow \{0,1\}^r$, sets $\mathsf{ct}_k = \mathsf{PKE.enc}(\mathrm{PK}, 1; r_k)$ and $v_k = (\mathsf{ct}_k, k)$.
   It computes a punctured PRF key $K\{(v_{k-1}, m_{k-1})\} \leftarrow F.\mathsf{puncture}(K, (v_{k-1}, m_{k-1}))$.
   It sets $\mathrm{PP} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}Enforce}\{k, v_{k-1}, m_{k-1}, K\{(v_{k-1}, m_{k-1})\}, \mathsf{ct}_k, \mathrm{PK}\})$ and sends (PP, v_0) to $\mathcal{A}$.
4. $\mathcal{A}$ sends a bit b'.

Claim 3.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ outputs } 0 \text{ in } \mathsf{Hyb}_0] - \Pr[\mathcal{A} \text{ outputs } 0 \text{ in } \mathsf{Hyb}_1] \leq \mathrm{negl}(\lambda).$$

Proof. Here, the behavior of Prog and Prog-Enforce are identical, as the hardwired value ct_k for Prog-Enforce is computed precisely as it is computed in Prog. ∎

Claim 3.2. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ outputs } 0 \text{ in } \mathsf{Hyb}_1] - \Pr[\mathcal{A} \text{ outputs } 0 \text{ in } \mathsf{Hyb}_2] \leq \mathrm{negl}(\lambda).$$

Proof. Here, the only difference is that the value of the F at the punctured point is replaced by a fresh random string. Since $\mathcal{A}$ is only receiving PP formed from a punctured key, this follows immediately from the selective security of F. ∎

Claim 3.3. Assuming $\mathcal{PKE}$ is an $\mathsf{IND\text{-}CPA}$ secure public key encryption scheme, for any PPT adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ outputs } 0 \text{ in } \mathsf{Hyb}_2] - \Pr[\mathcal{A} \text{ outputs } 0 \text{ in } \mathsf{Hyb}_3] \leq \text{negl}(\lambda).$$

Proof. The only difference between these two hybrids is that ct_k changes from a fresh encryption of 0 to a fresh encryption of 1. Hence this follows immediately from the $\mathsf{IND\text{-}CPA}$ security of $\mathcal{PKE}$. ■

In summary, it follows that if $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, and $\mathcal{PKE}$ is $\mathsf{IND\text{-}CPA}$ secure, then any PPT adversary $\mathcal{A}$ has negligible advantage in $\mathsf{Exp\text{-}Setup\text{-}Itr}(1^\lambda, \mathcal{I}, \mathcal{A})$. ■

Lemma 3.2 (Enforcing)**.** Assuming $\mathcal{PKE}$ is a perfectly correct public key encryption scheme, $\mathcal{I} = (\mathsf{Setup\text{-}Itr}, \mathsf{Setup\text{-}Itr\text{-}Enforce},$ is enforcing.

Proof. This follows immediately from perfect correctness of $\mathcal{PKE}$ because the hardwired value ct_k in $\mathsf{Prog\text{-}Enforce}$ is an encryption of 1, and all other states produced by $\mathsf{Prog\text{-}Enforce}$ are encryptions of 0. ■

4 Positional Accumulators

We will now define the notion of *positional accumulators*, and then show a construction based on $i\mathcal{O}$, puncturable PRFs, and public key encryption.

Intuitively, a positional accumulator will be a cryptographic data structure that maintains two values: a storage value and an accumulator value. The storage value will be allowed to grow comparatively large, while the accumulator value will be constrained to be short. Messages can be written to various positions in the the underlying storage, and new accumulated values can be computed as a stream, knowing only the previous accumulator value and the newly written message and its position in the data structure. Since the accumulator values are small, one cannot hope to recover everything written in the storage from the accumulator value alone. However, we define "helper" algorithms that essentially allow a party who is maintaining the full storage to help a more restricted party who is only maintaining the accumulator values recover the data currently written at an arbitrary location. The helper is not necessarily trusted, so the party maintaining the accumulator values performs a verification procedure in order to be convinced that they are indeed reading the correct messages.

A positional accumulator for message space $\mathcal{M}_\lambda$ consists of the following algorithms.

$\mathsf{Setup\text{-}Acc}(1^\lambda, T) \to \text{PP}, w_0, \textbf{store}_0$ 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 w_0, and an initial storage value $\textsc{store}_0$.

$\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, (m_1, \textbf{index}_1), \ldots, (m_k, \textbf{index}_k), \textbf{index}^*) \to \text{PP}, w_0, \textbf{store}_0$ 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 a sequence of symbol, index pairs, where each index is between 0 and $T-1$, and an additional $\textsc{index}^*$ also between 0 and $T-1$. It outputs public parameters PP, an initial accumulator value w_0, and an initial storage value $\textsc{store}_0$.

$\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}(1^\lambda, T, (m_1, \textbf{index}_1), \ldots, (m_k, \textbf{index}_k)) \to \text{PP}, w_0, \textbf{store}_0$ 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 w_0, and an initial storage value $\textsc{store}_0$.

$\mathsf{Prep\text{-}Read}(\text{PP}, \textbf{store}_{in}, \textbf{index}) \to m, \pi$ The prep-read algorithm takes as input the public parameters PP, a storage value $\textsc{store}_{in}$, and an index between 0 and $T-1$. It outputs a symbol m (that can be ϵ) and a value π.

$\mathsf{Prep\text{-}Write}(\mathrm{PP}, \mathbf{store}_{in}, \mathbf{index}) \rightarrow aux$ 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.

$\mathsf{Verify\text{-}Read}(\mathrm{PP}, w_{in}, m_{read}, \mathbf{index}, \pi) \rightarrow \{True, False\}$ The verify-read algorithm takes as input the public parameters PP, an accumulator value w_{in}, a symbol, m_{read}, an index between 0 and $T-1$, and a value π. It outputs $True$ or $False$.

$\mathsf{Write\text{-}Store}(\mathrm{PP}, \mathbf{store}_{in}, \mathbf{index}, m) \rightarrow \mathbf{store}_{out}$ The write-store algorithm takes in the public parameters, a storage value STORE_{in}, an index between 0 and $T-1$, and a symbol m. It outputs a storage value STORE_{out}.

$\mathsf{Update}(\mathrm{PP}, w_{in}, m_{write}, \mathbf{index}, aux) \rightarrow w_{out}$ **or** $Reject$ The update algorithm takes in the public parameters PP, an accumulator value w_{in}, a symbol m_{write}, and index between 0 and $T-1$, and an auxiliary value aux. It outputs an accumulator value w_{out} or $Reject$.

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

Correctness We consider any sequence $(m_1, \text{INDEX}_1), \ldots, (m_k, \text{INDEX}_k)$ of symbols $m_1, \ldots, m_k$ and indices $\text{INDEX}_1, \ldots, \text{INDEX}_k$ each between 0 and $T-1$. We fix any $\mathrm{PP}, w_0, \text{STORE}_0 \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. For j from 1 to k, we define STORE_j iteratively as $\text{STORE}_j := \mathsf{Write\text{-}Store}(\mathrm{PP}, \text{STORE}_{j-1}, \text{INDEX}_j, m_j)$. We similarly define aux_j and w_j iteratively as $aux_j := \mathsf{Prep\text{-}Write}(\mathrm{PP}, \text{STORE}_{j-1}, \text{INDEX}_j)$ and $w_j := Update(\mathrm{PP}, w_{j-1}, m_j, \text{INDEX}_j, aux_j)$. Note that the algorithms other than Setup-Acc are deterministic, so these definitions fix precise values, not random values (conditioned on the fixed starting values $\mathrm{PP}, w_0, \text{STORE}_0$).

We require the following correctness properties:

1. For every INDEX between 0 and $T-1$, $\mathsf{Prep\text{-}Read}(\mathrm{PP}, \text{STORE}_k, \text{INDEX})$ 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 $(m, \pi) \leftarrow \mathsf{Prep\text{-}Read}(\mathrm{PP}, \text{STORE}_k, \text{INDEX})$. Then $\mathsf{Verify\text{-}Read}(\mathrm{PP}, w_k, m, \text{INDEX}, \pi) = True$.

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 4.1 (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 $\mathsf{Exp\text{-}Setup\text{-}Acc}(1^\lambda, \mathsf{Acc}, \mathcal{A})$ is at most negligible in λ, where Exp-Setup-Acc is defined as follows.

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

1. Adversary chooses a bound $T \in \Theta(2^\lambda)$ and sends it to challenger.
2. $\mathcal{A}$ sends k messages $m_1, \ldots, m_k \in \mathcal{M}$ and k indices $\text{INDEX}_1, \ldots,$
$indexA_k \in \{0, \ldots, T-1\}$ to the challenger.
3. The challenger chooses a bit b. If $b = 0$, the challenger outputs $(\mathrm{PP}, w_0, \text{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. Else, it outputs $(\mathrm{PP}, w_0, \text{STORE}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, (m_1, \text{INDEX}_1), \ldots, (m_k, \text{INDEX}_k))$.
4. $\mathcal{A}$ sends a bit b'.

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

Definition 4.2 (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 $\mathsf{Exp\text{-}Setup\text{-}Acc}(1^\lambda, \mathsf{Acc}, \mathcal{A})$ is at most negligible in λ, where $\mathsf{Exp\text{-}Setup\text{-}Acc}$ is defined as follows.

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

1. Adversary chooses a bound $T \in \Theta(2^\lambda)$ and sends it to challenger.
2. $\mathcal{A}$ sends k messages $m_1, \ldots, m_k \in \mathcal{M}$ and k indices $\textsc{index}_1, \ldots,$ $indexA_k \in \{0, \ldots, T-1\}$ to the challenger.
3. The challenger chooses a bit b. If $b = 0$, the challenger outputs $(\mathrm{PP}, w_0, \textsc{store}_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. Else, it outputs $(\mathrm{PP}, w_0, \textsc{store}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}(1^\lambda, T, (m_1, \textsc{index}_1), \ldots, (m_k, \textsc{index}_k))$.
4. $\mathcal{A}$ sends a bit b'.

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

Definition 4.3 (Read Enforcing)**.** Consider any $\lambda \in \mathbb{N}$, $T \in \Theta(2^\lambda)$, $m_1, \ldots, m_k \in \mathcal{M}$, $\textsc{index}_1, \ldots, \textsc{index}_k \in \{0, \ldots, T-1\}$ and any $\textsc{index}^* \in \{0, \ldots, T-1\}$.

Let $(\mathrm{PP}, w_0, \mathsf{st}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, (m_1, \textsc{index}_1), \ldots, (m_k, \textsc{index}_k), \textsc{index}^*)$. For j from 1 to k, we define $\textsc{store}_j$ iteratively as $\textsc{store}_j := \mathsf{Write\text{-}Store}(\mathrm{PP}, \textsc{store}_{j-1}, \textsc{index}_j, m_j)$. We similarly define aux_j and w_j iteratively as $aux_j := \mathsf{Prep\text{-}Write}(\mathrm{PP}, \textsc{store}_{j-1}, \textsc{index}_j)$ and $w_j := Update(\mathrm{PP}, w_{j-1}, m_j, \textsc{index}_j, aux_j)$. Acc is said to be *read enforcing* if $\mathsf{Verify\text{-}Read}(\mathrm{PP}, w_k, m, \textsc{index}^*, \pi) = True$, then either $\textsc{index}^* \notin \{\textsc{index}_1, \ldots, \textsc{index}_k\}$ and $m = \epsilon$, or $m = m_i$ for the largest $i \in [k]$ such that $\textsc{index}_i = \textsc{index}^*$. Note that this is an information-theoretic property: we are requiring that for all other symobls m, values of π that would cause $\mathsf{Verify\text{-}Read}$ to output $True$ at $\textsc{index}^*$ do no exist.

Definition 4.4 (Write Enforcing)**.** Consider any $\lambda \in \mathbb{N}$, $T \in \Theta(2^\lambda)$, $m_1, \ldots, m_k \in \mathcal{M}$, $\textsc{index}_1, \ldots, \textsc{index}_k \in \{0, \ldots, T-1\}$. Let $(\mathrm{PP}, w_0, \mathsf{st}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}(1^\lambda, T, (m_1, \textsc{index}_1), \ldots, (m_k, \textsc{index}_k))$. For j from 1 to k, we define $\textsc{store}_j$ iteratively as $\textsc{store}_j := \mathsf{Write\text{-}Store}(\mathrm{PP}, \textsc{store}_{j-1}, \textsc{index}_j, m_j)$. We similarly define aux_j and w_j iteratively as $aux_j := \mathsf{Prep\text{-}Write}(\mathrm{PP}, \textsc{store}_{j-1}, \textsc{index}_j)$ and $w_j := Update(\mathrm{PP}, w_{j-1}, m_j, \textsc{index}_j, aux_j)$. Acc is said to be *write enforcing* if $\mathsf{Update}(\mathrm{PP}, w_{k-1}, m_k, \textsc{index}_k, aux) = w_{out} \neq Reject$, for any aux, then $w_{out} = w_k$. Note that this is an information-theoretic property: we are requiring that an aux value producing an accumulated value other than w_k or $Reject$ deos not exist.

4.1 Construction

In this section, we will construct a positional accumulator 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) for the symbol set $\mathcal{M} := \{0,1\}^{\ell'}$. Let $i\mathcal{O}$ be an indistinguishability obfuscator and let F be a selectively secure puncturable PRF with key space $\mathcal{K}$, punctured key space $\mathcal{K}_p$, domain $\{0,1\}^{\leq 2\ell+d}$, range $\{0,1\}^z$, and algorithms $F.\mathsf{setup}$, $F.\mathsf{puncture}$, and $F.\mathsf{eval}$. We let $\mathcal{PKE} = (\mathsf{PKE.setup}, \mathsf{PKE.enc}, \mathsf{PKE.dec})$ denote a public key encryption scheme with message space $\{0,1\}$ that is perfectly correct and uses z bits of randomness for encryption. We set ℓ to be sufficiently large so that ciphertext produces by $\mathcal{PKE}$, the special sympbol $\perp$, and all symbols in $\mathcal{M}$ are represented as unique ℓ-bit strings.

Without loss of generality, we assume that $T = 2^d$ for some integer $d = poly(\lambda)$. Our storage will take the form of a binary tree containing up to T leaves, with each node indexed by a binary string of length $\leq d$. A node in the tree indexed by a binary string $x_1x_2\ldots x_j \in \{0,1\}^{\leq d}$ is a child of the node indexed by $x_1 \ldots x_{j-1}$. An internal node contains an ℓ-bit string, as well as pointers to its left and right children. If either child does not exist, the pointer takes a default value $\perp$. A leaf node contains a symbol $\in \mathcal{M}$. The accumulated values will correspond to the ℓ bit strings stored at the root node.

- $\mathsf{Setup\text{-}Acc}(1^\lambda, T)$ The setup algorithm chooses a random key K for the puncturable PRF, and a secret key, public key pair $\mathrm{SK}, \mathrm{PK} \leftarrow \mathsf{PKE.setup}$. We let H denote the program in Figure 8.

Program H

Constants: Puncturable PRF key K, $\mathcal{PKE}$ public key PK.
Input: $h_1 \in \{0,1\}^\ell, h_2 \in \{0,1\}^\ell, \textsc{index} \in \{0,1\}^{<d}$

1. Compute $r = F(K, (h_1, h_2, \textsc{index}))$.
2. Output $\mathsf{PKE.enc}(\mathrm{PK}, 0; r)$.

Figure 3: Program H

The public parameters are $\mathrm{PP} := i\mathcal{O}(H)$. The initial value w_0 is $\perp$, and the initial $\textsc{store}_0$ is a single root node with value $\perp$.

- $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, (m_1, \textsc{index}_1), \ldots, (m_k, \textsc{index}_k), \textsc{index}^*) \rightarrow \mathrm{PP}, w_0, \textsc{store}_0$ The setup algorithm chooses a random key K for the puncturable PRF, and a secret key, public key pair $\mathrm{SK}, \mathrm{PK} \leftarrow \mathsf{PKE.setup}$. It creates ciphertexts $\mathrm{ct}_0, \ldots, \mathrm{ct}_{d-1}$ by encrypting 1 using $\mathsf{PKE.enc}$ with d independently sampled values for the randomness. For notational convenience, we also define $\mathrm{ct}_d := m_{i^*}$ where i^* is the largest integer such that $i^* = \textsc{index}^*$. If no such i^* exists, $\mathrm{ct}_d := \epsilon$.

  It defines $\textsc{store}_0$ to be a single root node with value $\perp$ and $w_0 := \perp$. It then defines the program H as in $\mathsf{Setup\text{-}Acc}$ above, and iteratively runs $\mathsf{Write\text{-}Store}(H, \textsc{store}_{i-1}, \textsc{index}_i, m_i) \rightarrow \textsc{store}_i$ to produce $\textsc{store}_k$. We let $\textsc{index}^*_j$ denote the j-bit prefix of $\textsc{index}$.

  We let H' denote the program in Figure 8.

Program H'

Constants: Puncturable PRF key K, $\mathcal{PKE}$ public key PK, $\textsc{index}^*$, values of nodes in $\textsc{store}_k$ for prefixes of $\textsc{index}^*$ and their siblings, $\{\mathrm{ct}_j\}_{j=0}^d$.
Input: $h_1 \in \{0,1\}^\ell, h_2 \in \{0,1\}^\ell, \textsc{index} \in \{0,1\}^{<d}$

1. If $\textsc{index}$ is a prefix of $\textsc{index}^*$ of length $j < d$, check if the value among $\{h_1, h_2\}$ corresponding to the child that is a length $j+1$ prefix of $\textsc{index}$ is equal to ct_{j+1}, and the other value among $\{h_1, h_2\}$ is checked against the corresponding node value in $\textsc{store}_k$. If both of these comparisons yield equality, output ct_j.
2. Otherwise, compute $r = F(K, (h_1, h_2, \textsc{index}))$ and output $\mathsf{PKE.enc}(\mathrm{PK}, 0; r)$.

Figure 4: Program H'

The public parameters are $\mathrm{PP} := i\mathcal{O}(H')$.

- $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}(1^\lambda, T, (m_1, \textsc{index}_1), \ldots, (m_k, \textsc{index}_k)) \rightarrow \mathrm{PP}, w_0, \textsc{store}_0$ This algorithm simply calls $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, (m_1, \textsc{index}_1), \ldots, (m_k, \textsc{index}_k), \textsc{index}^* = \textsc{index}_k)$.

- $\mathsf{Prep\text{-}Read}(\mathrm{PP}, \textsc{store}_{in}, \textsc{index}) \rightarrow m, \pi$ Here, $\textsc{index}$ is interpreted as a d-bit string. We let $j \in \{1, \ldots, d\}$ denote the largest value such that the node in $\textsc{store}_{in}$ corresponding to the length j prefix of $\textsc{index}$ is defined (i.e. not $\perp$). If $j = d$, then the leaf node corresponding to $\textsc{index}$ is defined, and contains some symbol which is assigned to m. Otherwise, $m := \epsilon$. π is formed as an ordered tuple of the ℓ-bit strings stored from the root down the path to the ancestor of $\textsc{index}$ at level j, as well as the strings stored at each sibling of a node along this path.

- $\mathsf{Prep\text{-}Write}(\mathrm{PP}, \textsc{store}_{in}, \textsc{index}) \rightarrow aux$ Similarly to $\mathsf{Prep\text{-}Read}$, we interpret $\textsc{index}$ as a d-bit string, and let $j \in \{1, \ldots, d\}$ be the largest value such that the node in $\textsc{store}_{in}$ corresponding to the length j prefix of $\textsc{index}$ is defined. aux is formed as ordered tuple of the ℓ-bit strings stored from the root down the path to the ancestor of $\textsc{index}$ at level j, as well as the strings stored at each sibling of these nodes.

- $\mathsf{Verify\text{-}Read}(\mathrm{PP}, w_{in}, m_{read}, \textsc{index}, \pi) \rightarrow \{True, False\}$ If the value of j reflected in π is $< d$ and $m_{read} \neq \epsilon$, then the $\mathsf{Verify\text{-}Read}$ algorithm outputs $False$. Otherwise, for each level i from 0 to $j-1$, the algorithm defines $h_{1,i+1}$ and $h_{2,i+1}$ to the be ℓ-bit strings given in π for the two children on level $i+1$ of the node at level i on the path to $\textsc{index}$. It computes $\mathrm{PP}(h_{1,i+1}, h_{2,i+1}, \textsc{index}_i)$, where $\textsc{index}_i$ denotes the i-bit prefix of $\textsc{index}$. If this output does not equal the ℓ-bit string given in π for this node, it outputs $False$. Similarly, if the root value given in π does not match w_{in}, it outputs $False$. If all of these checks pass, it outputs $True$. Note that when $j = d$, the value m_{read} is one of the input ℓ-bit strings for the check at level $d-1$.

- $\mathsf{Write\text{-}Store}(\mathrm{PP}, \textsc{store}_{in}, \textsc{index}, m) \to \textsc{store}_{out}$ First, if there are any nodes corresponding to prefixes of $\textsc{index}$ or siblings of prefixes of $\textsc{index}$ that are not yet initialized in $\textsc{store}_{in}$, they are initialized with $\perp$ values. Next, the ℓ-bit string associated with the node $\textsc{index}$ is set to m. For each i from 0 to $d-1$, we define $h_{1,i+1}, h_{2,i+1}$ denote the ℓ-bit strings (or $\perp$) values associated with the two children of the node corresponding to the i-bit prefix of $\textsc{index}$. We iteratively update the values for the prefixes of $\textsc{index}$, starting with $i = d-1$ and setting the value at the i-bit prefix equal to $\mathrm{PP}(h_{1,i}, h_{2,i}, \textsc{index}_i)$ (again, $\textsc{index}_i$ denotes the i-bit prefix of $\textsc{index}$). After these changes are made all the way up through the root, the resulting tree is output as $\textsc{store}_{out}$.

- $\mathsf{Update}(\mathrm{PP}, w_{in}, m_{write}, \textsc{index}, aux) \to w_{out}$ or $Reject$ The update algorithm first performs the same checks as described in the Verify-Read algorithm. If any of these fail, it outs $Reject$. Otherwise, using the values in aux, it re-computes the ℓ-bit values for the nodes whose prefixes of $\textsc{index}$ as described in the Write-Store algorithm, starting by replacing the value at $\textsc{index}$ itself with m_{write} (note that the values given in aux suffice for this). It outputs the new root value as w_{out}.

4.1.1 Correctness

We consider any sequence $(m_1, \textsc{index}_1), \ldots, (m_k, \textsc{index}_k)$ of symbols $m_1, \ldots, m_k$ and indices $\textsc{index}_1, \ldots, \textsc{index}_k$ each between 0 and $T-1$. We fix any $\mathrm{PP}, w_0, \textsc{store}_0 \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. For j from 1 to k, we define $\textsc{store}_j$ iteratively as $\textsc{store}_j := \mathsf{Write\text{-}Store}(\mathrm{PP}, \textsc{store}_{j-1}, \textsc{index}_j, m_j)$. We similarly define aux_j and w_j iteratively as $aux_j := \mathsf{Prep\text{-}Write}(\mathrm{PP}, \textsc{store}_{j-1}, \textsc{index}_j)$ and $w_j := Update(\mathrm{PP}, w_{j-1}, m_j, \textsc{index}_j, aux_j)$.

By definition of the algorithm Write-Store, it is immediate that a leaf of $\textsc{store}_k$ corresponding to any $\textsc{index}$ contains the value m_i for the largest i such that $\textsc{index}_i = \textsc{index}$ if such an i exists, otherwise it is uninitialized or has the $\perp$ value. The Prep-Read algorithm therefore correctly returns m_i or ϵ in these cases respectively. This ensures correctness property 1.

To see that correctness property 2 also holds, we observe that every call of Write-Store maintains the invariant that for any initialized internal node (at some $\textsc{index}$) the ℓ-bit string that it stores is equal to $\mathrm{PP}(h_1, h_2, \textsc{index})$, where h_1 and h_2 are the values stored at its children. Thus, the checks performed by Verify-Read will pass because the output path produced by Prep-Read maintains this invariant.

4.1.2 Security

We now prove our construction above satisfies indistinguishability of read and write setup:

Lemma 4.1. Assuming that F is a selectively secure puncturable PRF and $i\mathcal{O}$ is an indistinguishability obfuscator, our construction satisfies indistinguishability of read setup.

Proof. We let $\mathsf{Exp\text{-}Setup\text{-}Acc}_0$ denote the version of the security experiment where $b = 0$ (i.e. we are running the real Setup-Acc algorithm), and $\mathsf{Exp\text{-}Setup\text{-}Acc}_1$ denote the version of the security experiment where $b = 1$ (i.e. we are running the Setup-Acc-Enforce-Read algorithm). We will show these two experiments are computationally indistinguishably using a hybrid argument that gradually transitions from $\mathsf{Exp\text{-}Setup\text{-}Acc}_0$ to $\mathsf{Exp\text{-}Setup\text{-}Acc}_1$.

Our intermediary experiments will be parameterized by a depth parameter z between 0 and $d-1$. In all experiments, the challenger will choose $K, \mathrm{PK}, \mathrm{SK}$ as is common to the Setup-Acc, Setup-Acc-Enforce-Read algorithms, and will define $\mathrm{ct}_0, \ldots, \mathrm{ct}_d, \textsc{store}_k$ as in the Setup-Acc-Enforce-Read algorithm. The experiments will differ only in the program to be obfuscated to form the public parameters.

We first define:

$\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.1,z}$ In this experiment, the program to be obfuscated is defined as follows. We first set $r^* := F(K, h_{1,z}, h_{2,z}, \text{INDEX}^*_z)$, where INDEX^*_z denotes the length z prefix of INDEX^*, whichever of $\{h_{1,z}, h_{2,z}\}$ corresponds to the length $z+1$ prefix of INDEX^* is set to be C_{z+1}, and the other is set equal to the corresponding value in STORE_k. We also compute $K\{(h_{1,z}, h_{2,z}, \text{INDEX}^*z)\} \leftarrow F.\mathsf{puncture}(K, (h_{1,z}, h_{2,z}, \text{INDEX}^*z))$.

Program $H_{0.1,z}$

Constants: Punctured PRF key $K\{(h_{1,z}, h_{2,z}, \text{INDEX}^*z)\}$, $\mathcal{PKE}$ public key PK, randomness r^*, $h_{1,z}$, $h_{2,z}$, INDEX^*, values of nodes in STORE_k for prefixes of INDEX^* and their siblings, $\{\text{ct}_j\}_{j=0}^d$.
Input: $h_1 \in \{0,1\}^\ell, h_2 \in \{0,1\}^\ell, \text{INDEX} \in \{0,1\}^{<d}$

1. If INDEX is a prefix of INDEX^* of length $z < j < d$, check if the value among $\{h_1, h_2\}$ corresponding to the child that is a length $j+1$ prefix of INDEX is equal to ct_{j+1}, and the other value among $\{h_1, h_2\}$ is checked against the corresponding node value in STORE_k. If both of these comparisons yield equality, output ct_j.
2. If INDEX is a prefix of INDEX^* of length z, $h_1 = h_{1,z}$, and $h_2 = h_{2,z}$, output $\mathsf{PKE.enc}(\text{PK}, 0; r^*)$.
3. Otherwise, compute $r = F(K, (h_1, h_2, \text{INDEX}))$ and output $\mathsf{PKE.enc}(\text{PK}, 0; r)$.

Figure 5: Program $H_{0.1,z}$

We next define:

$\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.2,z}$ In this experiment, the program to be obfuscated is defined as follows. We again let INDEX^*_z denote the length z prefix of INDEX^*, whichever of $\{h_{1,z}, h_{2,z}\}$ corresponds to the length $z+1$ prefix of INDEX^* is set to be C_{z+1}, and the other is set equal to the corresponding value in STORE_k. We also compute $K\{(h_{1,z}, h_{2,z}, \text{INDEX}^*z)\} \leftarrow F.\mathsf{puncture}(K, (h_{1,z}, h_{2,z}, \text{INDEX}^*z))$. We set r^* to be a fresh random string.

Program $H_{0.2,z}$

Constants: Punctured PRF key $K\{(h_{1,z}, h_{2,z}, \text{INDEX}^*z)\}$, $\mathcal{PKE}$ public key PK, randomness r^*, $h_{1,z}$, $h_{2,z}$, INDEX^*, values of nodes in STORE_k for prefixes of INDEX^* and their siblings, $\{\text{ct}_j\}_{j=0}^d$.
Input: $h_1 \in \{0,1\}^\ell, h_2 \in \{0,1\}^\ell, \text{INDEX} \in \{0,1\}^{<d}$

1. If INDEX is a prefix of INDEX^* of length $z < j < d$, check if the value among $\{h_1, h_2\}$ corresponding to the child that is a length $j+1$ prefix of INDEX is equal to ct_{j+1}, and the other value among $\{h_1, h_2\}$ is checked against the corresponding node value in STORE_k. If both of these comparisons yield equality, output ct_j.
2. If INDEX is a prefix of INDEX^* of length z, $h_1 = h_{1,z}$, and $h_2 = h_{2,z}$, output $\mathsf{PKE.enc}(\text{PK}, 0; r^*)$.
3. Otherwise, compute $r = F(K, (h_1, h_2, \text{INDEX}))$ and output $\mathsf{PKE.enc}(\text{PK}, 0; r)$.

Figure 6: Program $H_{0.2,z}$

We then define:

$\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.3,z}$ In this experiment, the program to be obfuscated is defined as follows. We again let INDEX^*_z denote the length z prefix of INDEX^*, whichever of $\{h_{1,z}, h_{2,z}\}$ corresponds to the length $z+1$ prefix of INDEX^* is set to be C_{z+1}, and the other is set equal to the corresponding value in STORE_k. We also compute $K\{(h_{1,z}, h_{2,z}, \text{INDEX}^*z)\} \leftarrow F.\mathsf{puncture}(K, (h_{1,z}, h_{2,z}, \text{INDEX}^*z))$. We set r^* to be a fresh random string.

Lastly, we define:

$\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.4,z}$ In this experiment, the program to be obfuscated is defined as follows. We again let INDEX^*_z denote the length z prefix of INDEX^*, whichever of $\{h_{1,z}, h_{2,z}\}$ corresponds to the length $z+1$ prefix of INDEX^* is set to be C_{z+1}, and the other is set equal to the corresponding value in STORE_k. We set r^* to be a fresh random string.

Program $H_{0.3,z}$

Constants: Punctured PRF key $K\{(h_{1,z}, h_{2,z}, \text{INDEX}^*z)\}$, $\mathcal{PKE}$ public key PK, randomness r^*, $h_{1,z}$, $h_{2,z}$, INDEX^*, values of nodes in STORE_k for prefixes of INDEX^* and their siblings, $\{\text{ct}_j\}_{j=0}^d$.
Input: $h_1 \in \{0,1\}^\ell, h_2 \in \{0,1\}^\ell, \text{INDEX} \in \{0,1\}^{<d}$

1. If INDEX is a prefix of INDEX^* of length $z < j < d$, check if the value among $\{h_1, h_2\}$ corresponding to the child that is a length $j+1$ prefix of INDEX is equal to ct_{j+1}, and the other value among $\{h_1, h_2\}$ is checked against the corresponding node value in STORE_k. If both of these comparisons yield equality, output ct_j.
2. If INDEX is a prefix of INDEX^* of length z, $h_1 = h_{1,z}$, and $h_2 = h_{2,z}$, output $\mathsf{PKE.enc}(\text{PK}, 1; r^*)$.
3. Otherwise, compute $r = F(K, (h_1, h_2, \text{INDEX}))$ and output $\mathsf{PKE.enc}(\text{PK}, 0; r)$.

Figure 7: Program $H_{0.3,z}$

Program $H_{0.4,z}$

Constants: Punctured PRF key K, $\mathcal{PKE}$ public key PK, randomness r^*, $h_{1,z}$, $h_{2,z}$, INDEX^*, values of nodes in STORE_k for prefixes of INDEX^* and their siblings, $\{\text{ct}_j\}_{j=0}^d$.
Input: $h_1 \in \{0,1\}^\ell, h_2 \in \{0,1\}^\ell, \text{INDEX} \in \{0,1\}^{<d}$

1. If INDEX is a prefix of INDEX^* of length $z < j < d$, check if the value among $\{h_1, h_2\}$ corresponding to the child that is a length $j+1$ prefix of INDEX is equal to ct_{j+1}, and the other value among $\{h_1, h_2\}$ is checked against the corresponding node value in STORE_k. If both of these comparisons yield equality, output ct_j.
2. If INDEX is a prefix of INDEX^* of length z, $h_1 = h_{1,z}$, and $h_2 = h_{2,z}$, output $\mathsf{PKE.enc}(\text{PK}, 1; r^*)$.
3. Otherwise, compute $r = F(K, (h_1, h_2, \text{INDEX}))$ and output $\mathsf{PKE.enc}(\text{PK}, 0; r)$.

Figure 8: Program $H_{0.4,z}$

Now that we have these experiments defined, we will argue that we can start with $\mathsf{Exp\text{-}Setup\text{-}Acc}_0$, transition to $\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.1,d-1}$, then to $\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.2,d-1}$, then to $\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.3,d-1}$, then to $\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.4,d-1}$, then to $\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.1,d-2}$, and so on. We finally arrive at $\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.4,0}$, which is identical to $\mathsf{Exp\text{-}Setup\text{-}Acc}_1$.

We make the following claims for each z from $d-1$ to 0. For notational convenience, we consider $\mathsf{Exp\text{-}Setup\text{-}Acc}_{0.4,d}$ to be another name for $\mathsf{Exp\text{-}Setup\text{-}Acc}_0$.

Claim 4.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ outputs 0 in } \mathsf{Exp\text{-}Setup\text{-}Acc}_{0.4,z+1}] - \Pr[\mathcal{A} \text{ outputs 0 in } \mathsf{Exp\text{-}Setup\text{-}Acc}_{0.1,z}] \leq \text{negl}(\lambda).$$

Proof. To see this, note that the hardwired value r^* in the program $H_{0.1,z}$ is the same that will be computed from the unpunctured key in the program $H_{0.4,z+1}$. We also note that the ciphertext produces in line 2. of $H_{0.4,z+1}$ is distributed identically to ct_{z+1}, so hardwired the randomness in $H_{0.4,z+1}$ and hard-coding ct_{z+1} in $H_{0.1,z}$ does not cause a difference. The input/output behavior of $H_{0.1,z}$ and $H_{0.4,z+1}$ are hence identical, so the $i\mathcal{O}$ security applies. ∎

Claim 4.2. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ outputs 0 in } \mathsf{Exp\text{-}Setup\text{-}Acc}_{0.1,z}] - \Pr[\mathcal{A} \text{ outputs 0 in } \mathsf{Exp\text{-}Setup\text{-}Acc}_{0.2,z}] \leq \text{negl}(\lambda).$$

Proof. The only change being made between $H_{0.1,z}$ and $H_{0.2,z}$ is that the hardwired output of F at the punctured point is replaced by a freshly random, hardwired r^* value. Computationally indistinguishability then follows immediately from the selective security of F. ∎

Claim 4.3. Assuming $\mathcal{PKE}$ is an IND-CPA secure public key encryption scheme, for any PPT adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ outputs 0 in } \mathsf{Exp\text{-}Setup\text{-}Acc}_{0.1,z}] - \Pr[\mathcal{A} \text{ outputs 0 in } \mathsf{Exp\text{-}Setup\text{-}Acc}_{0.2,z}] \leq \text{negl}(\lambda).$$

Proof. This follows immediately from the definition of CPA security, as the only change is that a (freshly random) encryption of 0 is replaced by a (freshly random) encryption of 1. ■

Claim 4.4. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ outputs } 0 \text{ in } \mathsf{Exp\text{-}Setup\text{-}Acc}_{0.3,z}] - \Pr[\mathcal{A} \text{ outputs } 0 \text{ in } \mathsf{Exp\text{-}Setup\text{-}Acc}_{0.4,z}] \leq \text{negl}(\lambda).$$

Proof. Here, the behavior of the programs $H_{0.3,z}$ and $H_{0.4,z}$ are identical, because the punctured point value is never computed. Hence the $i\mathcal{O}$ security guarantee applies. ■

■

Lemma 4.2. Assuming that F is a selectively secure puncturable PRF and $i\mathcal{O}$ is an indistinguishability obfuscator, our construction satisfies indistinguishability of write setup.

Proof. This follows immediately from Lemma 4.1, as the $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}$ algorithm simply calls an instance of the $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}$ algorithm. ■

We now establish that our construction satisfies the required information-theoretic enforcement properties:

Lemma 4.3. Our construction is Read Enforcing.

Proof. We consider a sequence $(m_1, \text{INDEX}_1), \ldots, (m_k, \text{INDEX}_k)$ and a INDEX^*, and we let $\text{PP}, w_0, \text{STORE}_0 \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, (m_1, \text{INDEX}_1), \ldots, (m_k, \text{INDEX}_k), \text{INDEX}^*)$. We let w_k, STORE_k denote the resulting accumulator and storage values after then storing and updating this sequence using PP, starting from w_0, STORE_0. The "right" m to be read at INDEX^* is m_i for the largest $i \in \{1, \ldots, k\}$ such that $\text{INDEX}_i = \text{INDEX}^*$, if such an i exists. Otherwise, $m := \epsilon$.

We suppose (for contradiction) that for some $\tilde{m} \neq m$, there exists a value $\tilde{\pi}$ such that $\mathsf{Verify\text{-}Read}(\text{PP}, w_k, \tilde{m}, \text{INDEX}^*, \tilde{\pi}) = True$. Now, by definition of the program H' that is obfuscated to form PP, w_k will be the value ct_0 (the value stored at the root of the tree in STORE_k). Since this is an encryption of 1 and the scheme $\mathcal{PKE}$ is perfectly correct, it will never be an output produced by line 2. of the program H. Hence, the only values for its children that will pass the check performed by $\mathsf{Verify\text{-}Read}$ are the unique values that are checked in line 1. of H'. Applying this reasoning iteratively down the tree, we see that the values in π must match the true values stored in the tree in STORE_k at these nodes, and hence this applies also at the leaf where m is stored (or where the path to INDEX^* reaches a $\perp$). Thus, we cannot have $\tilde{m} \neq m$ and still have a successful verification. ■

Lemma 4.4. Our construction is Write Enforcing.

Proof. As in the proof of Lemma 4.3, we have that the only value of aux that will pass the checks performed by the Update algorithm is the true values along the relevant path as stored in STORE_k. This will then deterministically produce w_k as the output of Update. ■

5 Splittable Signatures

In this section, we will define the syntax and correctness/security properties of *splittable signatures* and then show a construction based on indistinguishability obfuscation and pseudorandom generators.

A splittable signature scheme will essentially consist of a normal signature scheme, augmented by some additional algorithms that produce alternative signing and verification keys with differing capabilities. More precisely, there will be "all but one" keys that work correctly except for a single message m^*, and there will be "one" keys that work only for m^*. There will also be a reject-verification key that always outputs reject when used to verify a signature.

Our required security properties will be closer in spirit to the typical security properties of MACs as opposed to signatures, since we do not provide access to a signing oracle in our security games. Our properties are nonetheless sufficient for our application, and avoiding unnecessary signing oracles makes it possible to argue that these different kinds of verification keys are computationally indistinguishable, the attacker is not provided with a signature on which they behave differently.

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.

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 $m \in \mathcal{M}, m \neq m^*$, $\mathsf{Sign}(\mathrm{SK}, m) = \mathsf{Sign\text{-}Spl\text{-}abo}(\mathrm{SK}_{\mathrm{abo}}, m)$.
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 5.1 ($\mathrm{VK}_{\mathrm{rej}}$ 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 5.2 ($\mathrm{VK}_{\mathrm{one}}$ 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 5.3 ($\mathrm{VK}_{\mathrm{abo}}$ 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 5.4 (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 $\mathrm{SK}_{\mathrm{abo}}$.

5.1 Construction

Let $\mathcal{M}$ be the message space. For simplicity of notations, we will assume $\mathcal{M} = \{0,1\}^\ell$, where ℓ is polynomial in the security parameter λ. Let F be a puncturable pseudorandom function with key space $\mathcal{K}$, punctured key space $\mathcal{K}_p$, domain $\mathcal{M}$, range $\{0,1\}^\lambda$ and algorithms $F.\mathsf{setup}$, $F.\mathsf{puncture}$ and $F.\mathsf{eval}$. Finally, we will also use an indistinguishability obfuscator $i\mathcal{O}$ and an injective pseudorandom generator $\mathrm{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. (The pseudorandom generator will be used in the proof, but will not be needed in the actual scheme.) We will now define the algorithms Setup-Spl, Sign-Spl, Verify-Spl, Split and Sign-Spl-abo.

- Setup-Spl(1^λ) The setup algorithm takes as input security parameter λ and chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$. Next, it chooses $x \leftarrow \{0,1\}^\lambda$. The secret key SK is set to be (K_1, K_2, x), the verification key is an obfuscation of the program Prog-VK defined in Figure 9; $\mathrm{VK} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{VK})$, and $\mathrm{VK}_{\mathrm{rej}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{rej}})$, where $\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{rej}}$ is defined in Figure 10.

Prog-VK

Constants Puncturable PRF keys $K_1, K_2 \in \mathcal{K}$, $x \in \{0,1\}^\lambda$.
Inputs Message $m \in \mathcal{M}$, signature $\sigma = (\sigma_1, \sigma_2)$.

1. If $\sigma_1 = (F(K_1, m) \oplus x)$ **and** $\sigma_2 = F(K_2, m)$ output 1. Else output 0.

Figure 9: Prog-VK

$\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{rej}}$

Inputs: Message $m \in \mathcal{M}$, signature $\sigma = (\sigma_1, \sigma_2)$.

1. Output 0.

Figure 10: $\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{rej}}$

- Sign-Spl(SK, m) The signing algorithm takes as input the secret key $\mathrm{SK} = (K_1, K_2, x)$ and message $m \in \mathcal{M}$. It outputs $\sigma = (F(K_1, m) \oplus x, F(K_2, m))$.
- Verify-Spl(VK, m, σ) The verification algorithm simply executes the verification key with inputs m and σ. It outputs $\mathrm{VK}(m, \sigma)$.
- Split(SK, m^*) The splitting algorithm takes as input secret key $\mathrm{SK} = (K_1, K_2, x)$ and a message $m^* \in \mathcal{M}$. It computes $\sigma_{\mathrm{one}} = \mathsf{Sign}(\mathrm{SK}, m^*)$. Next, it computes $\mathrm{VK}_{\mathrm{one}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{one}})$, $\mathrm{SK}_{\mathrm{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{SK}_{\mathrm{abo}})$ and $\mathrm{VK}_{\mathrm{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{abo}})$, where the programs $\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{one}}$, $\mathsf{Prog}\text{-}\mathrm{SK}_{\mathrm{abo}}$ and $\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{abo}}$ are defined in Figures 11, 12 and 13 respectively.

$\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{one}}$

Constants Signature components $s_1, s_2 \in \{0,1\}^\lambda$, message $m^* \in \mathcal{M}$.
Inputs Message $m \in \mathcal{M}$, signature $\sigma = (\sigma_1, \sigma_2) \in \{0,1\}^{2\lambda}$.

1. If $m \neq m^*$ output 0.
2. If $\sigma_1 = s_1$ **and** $\sigma_2 = s_2$ output 1. Else output 0.

Figure 11: $\mathsf{Prog}\text{-}\mathrm{VK}_{\mathrm{one}}$

Correctness For correctness, note that Properties 1, 3, 5 and 6 follow directly from construction, while Properties 2 and 4 follow from the correctness of puncturable PRF keys.

5.1.1 Proofs of Security

We will now show that $\mathcal{S} = (\mathsf{Setup\text{-}Spl}, \mathsf{Sign\text{-}Spl}, \mathsf{Verify\text{-}Spl}, \mathsf{Split}, \mathsf{Sign\text{-}Spl\text{-}abo})$ satisfies $\mathrm{VK}_{\mathrm{rej}}$ indistinguishability, $\mathrm{VK}_{\mathrm{one}}$ indistinguishability, $\mathrm{VK}_{\mathrm{abo}}$ indistinguishability and splitting indistinguishability.

Prog-SK$_{\text{abo}}$

Constants Message $m^* \in \mathcal{M}$, punctured PRF keys $K_1\{m^*\}, K_2\{m^*\} \in \mathcal{K}_p$, $x \in \{0,1\}^\lambda$.
Inputs Message $m \in \mathcal{M}$.

1. If $m = m^*$, output $\perp$.
2. Compute $\sigma_1 = F.\mathsf{eval}(K_1\{m^*\}, m) \oplus x$ and $\sigma_2 = F.\mathsf{eval}(K\{m^*\}, m)$. Output $\sigma = (\sigma_1, \sigma_2)$.

Figure 12: Prog-SK$_{\text{abo}}$

Prog-VK$_{\text{abo}}$

Constants Message $m^* \in \mathcal{M}$, punctured PRF keys $K_1\{m^*\}, K_2\{m^*\} \in \mathcal{K}$, $x \in \{0,1\}^\lambda$.
Inputs message $m \in \mathcal{M}$, signature $\sigma = (\sigma_1, \sigma_2)$.

1. If $m = m^*$, output 0.
2. If $\sigma_1 = F.\mathsf{eval}(K_1\{m^*\}, m) \oplus x$ **and** $\sigma_2 = F.\mathsf{eval}(K_2\{m^*\}, m)$ output 1. Else output 0.

Figure 13: Prog-VK$_{\text{abo}}$

Lemma 5.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and PRG an injective pseudorandom generator, any PPT adversary $\mathcal{A}$ has negligible advantage in $\mathsf{Exp}\text{-}\mathrm{VK}_{\text{rej}}(1^\lambda, \mathcal{S}, \mathcal{A})$.

Proof. We will define a sequence of hybrid experiments $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_3$, and then show that the outputs of each hybrid are computationally indistinguishable.

Hyb_0 In this experiment, the challenger sends VK to $\mathcal{A}$.

1. Challenger chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \leftarrow \{0,1\}^\lambda$. It sends $\mathrm{VK} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{VK}\{K_1, K_2, x\})$ to $\mathcal{A}$.

Hyb_1 In this experiment, the challenger outputs an obfuscation of Prog-VK$'_{\text{rej}}$ (defined in Figure 14).

1. Challenger chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \leftarrow \{0,1\}^\lambda$. It computes $y = \mathrm{PRG}(x)$ sends $\mathrm{VK} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{VK}'_{\text{rej}}\{K_1, K_2, y\})$ to $\mathcal{A}$.

Prog-VK$'_{\text{rej}}$

Constants Puncturable PRF keys $K_1, K_2 \in \mathcal{K}$, $y \in \{0,1\}^{2\lambda}$.
Inputs Message $m \in \mathcal{M}$, signature $\sigma = (\sigma_1, \sigma_2)$.

1. If $y = \mathrm{PRG}(F(K_1, m) \oplus \sigma_1)$ **and** $\sigma_2 = F(K_2, m)$ output 1. Else output 0.

Figure 14: Prog-VK$'_{\text{rej}}$

Hyb_2 This experiment is similar to the previous one, except that the challenger chooses y as a uniformly random λ bit string.

1. Challenger chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$. It chooses $y \leftarrow \{0,1\}^{2\lambda}$ and sends $\mathrm{VK} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{VK}'_{\text{rej}}\{K_1, K_2, y\})$ to $\mathcal{A}$.

Hyb_3 In this experiment, the challenger outputs an obfuscation of Prog-VK$_{\text{rej}}$.

1. Challenger sends $\mathrm{VK} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\mathrm{VK}_{\text{rej}})$ to $\mathcal{A}$.

Analysis Let $p_{\mathcal{A},i}$ denote the probability that $\mathcal{A}$ outputs 1 in Hyb_i.

Claim 5.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|p_{\mathcal{A},0} - p_{\mathcal{A},1}| \leq \text{negl}(\lambda)$.

Proof. This follows from the fact that PRG is injective, and hence the programs $\mathsf{Prog\text{-}VK}\{K_1, K_2, x\}$ and $\mathsf{Prog\text{-}VK}'_{\text{rej}}\{K_1, K_2, y\}$ are functionally identical. As a result, their obfuscations are computationally indistinguishable by the $i\mathcal{O}$ security requirement. ∎

Claim 5.2. Assuming PRG is a secure pseudorandom generator, for any PPT $\mathcal{A}$, $|p_{\mathcal{A},1} - p_{\mathcal{A},2}| \leq \text{negl}(\lambda)$.

Proof. Now that the pre-image x is no longer referenced in the verification program, the fact that a random image y can be replaced by a uniformly random value in $\{0,1\}^{2\lambda}$ follows directly from the security of PRG. ∎

Claim 5.3. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|p_{\mathcal{A},2} - p_{\mathcal{A},3}| \leq \text{negl}(\lambda)$.

Proof. Note that since y is chosen uniformly at random in Hyb_2, with overwhelming probability, it does not lie in the range of PRG. As a result, $\mathsf{Prog\text{-}VK}'_{\text{rej}}$ outputs 0 on all inputs, making it functionally identical to $\mathsf{Prog\text{-}VK}_{\text{rej}}$. Therefore, their obfuscations are computationally indistinguishable by the $i\mathcal{O}$ security requirement. ∎

∎

Lemma 5.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F a secure puncturable PRF and PRG a secure pseudorandom generator, any PPT adversary $\mathcal{A}$ has negligible advantage in $\mathsf{Exp\text{-}VK}_{\text{one}}(1^\lambda, \mathcal{S}, \mathcal{A})$.

Proof. To prove this lemma, we will define a sequence of hybrid experiments $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_7$.

Hyb_0 In this experiment, the challenger computes $(\sigma_{\text{one}}, \text{VK})$ as in the original scheme.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1, K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $x \leftarrow \{0,1\}^\lambda$. It sets $s_1 = F(K_1, m^*) \oplus x$, $s_2 = F(K_2, m^*)$, $\sigma_{\text{one}} = (s_1, s_2)$, $\text{VK} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK}\{K_1 K_2, x\})$ and sends $(\sigma_{\text{one}}, \text{VK})$ to $\mathcal{A}$.

Hyb_1 This experiment is similar to the previous one, except that the challenger hardwires a punctured PRF key instead of K_1 in the verification key VK, and $y = \text{PRG}(x)$ is used to check instead of x itself:

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1, K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $x \leftarrow \{0,1\}^\lambda$. It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $s_1 = F(K_1, m^*) \oplus x, s_2 = F(K_2, m^*), \sigma_{\text{one}} = (s_1, s_2)$. It sets $y = \text{PRG}(x), \text{VK} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK\text{-}Alt}\{m^*, K_1\{m^*\}, K_2, y, \sigma_{\text{one}}\})$ ($\mathsf{Prog\text{-}VK\text{-}Alt}$ is defined in Figure 15) and sends $(\sigma_{\text{one}}, \text{VK})$ to $\mathcal{A}$.

Hyb_2 In this experiment, $F(K_1, m^*)$ is set to be a uniformly random string in $\{0,1\}^\lambda$.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1, K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $x \leftarrow \{0,1\}^\lambda$. It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, chooses $r \leftarrow \{0,1\}^\lambda$ and sets $s_1 = r \oplus x$, $s_2 = F(K_2, m^*)$, $\sigma_{\text{one}} = (s_1, s_2)$. It sets $y = \text{PRG}(x), \text{VK} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK\text{-}Alt}\{m^*, K_1\{m^*\}, K_2, y, \sigma_{\text{one}}\})$ and sends $(\sigma_{\text{one}}, \text{VK})$ to $\mathcal{A}$.

Prog-VK-Alt
Constants Message $m^* \in \mathcal{M}$, punctured PRF key $K_1\{m^*\} \in \mathcal{K}_p$, PRF key $K_2 \in \mathcal{K}$, $y \in \{0,1\}^{2\lambda}$, $s \in \{0,1\}^\lambda \times \{0,1\}^\lambda$. **Inputs** Message $m \in \mathcal{M}$, Signature tuple $\sigma = (\sigma_1, \sigma_2) \in \{0,1\}^{2\lambda}$. 1. If $m \neq m^*$ (a) If $y = \text{PRG}((F.\mathsf{eval}(K_1\{m^*\}, m) \oplus \sigma_1))$ **and** $\sigma_2 = F(K_2, m)$ output 1. Else output 0. 2. Else if $m \neq m^*$ (a) If $s = (\sigma_1, \sigma_2)$ output 1. Else output 0.

Figure 15: Prog-VK-Alt

Hyb_3 In this experiment, s_1 is chosen uniformly at random from $\{0,1\}^\lambda$.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1, K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, chooses $s_1 \leftarrow \{0,1\}^\lambda$, sets $s_2 = F(K_2, m^*)$, $\sigma_{\text{one}} = (s_1, s_2)$.
   It sets $y = \text{PRG}(x), \text{VK} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK\text{-}Alt}\{m^*, K_1\{m^*\}, K_2, y, \sigma_{\text{one}}\})$ and sends $(\sigma_{\text{one}}, \text{VK})$ to $\mathcal{A}$.

Hyb_4 This experiment is identical to the previous one, except that the challenger sets y to be a uniformly random 2λ bit string, instead of $\text{PRG}(x)$.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1, K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, chooses $s_1 \leftarrow \{0,1\}^\lambda$, sets $s_2 = F(K_2, m^*)$, $\sigma_{\text{one}} = (s_1, s_2)$.
   It chooses $y \leftarrow \{0,1\}^{2\lambda}$, sets $\text{VK} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK\text{-}Alt}\{m^*, K_1\{m^*\}, K_2, y, \sigma_{\text{one}}\})$ and sends $(\sigma_{\text{one}}, \text{VK})$ to $\mathcal{A}$.

Hyb_5 In this experiment, the challenger outputs an obfuscation of $\mathsf{Prog\text{-}VK}_{\text{one}}$.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1, K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $x \leftarrow \{0,1\}^\lambda$.
   It chooses $s_1 \leftarrow \{0,1\}^\lambda$, sets $s_2 = F(K_2, m^*)$, $\sigma_{\text{one}} = (s_1, s_2)$.
   It sets $\text{VK} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK}_{\text{one}}\{m^*, \sigma_{\text{one}}\})$ and sends $(\sigma_{\text{one}}, \text{VK})$ to $\mathcal{A}$.

Hyb_6 This experiment is identical to the previous one, the only difference being a syntactical change. It chooses $r \leftarrow \{0,1\}^\lambda$, $x \leftarrow \{0,1\}^\lambda$ and sets $s_1 = r \oplus x$.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1, K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, chooses $r, x \leftarrow \{0,1\}^\lambda$ and sets $s_1 = r \oplus x$, $s_2 = F(K_2, m^*)$, $\sigma_{\text{one}} = (s_1, s_2)$.
   It sets $\text{VK} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK}_{\text{one}}\{m^*, \sigma_{\text{one}}\})$ and sends $(\sigma_{\text{one}}, \text{VK})$ to $\mathcal{A}$.

Hyb_7 In this experiment, the challenger sets $s_1 = F(K_1, m^*) \oplus x$.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1, K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, chooses $x \leftarrow \{0,1\}^\lambda$ and sets $s_1 = F(K_1, m^*) \oplus x$, $s_2 = F(K_2, m^*)$, $\sigma_{\text{one}} = (s_1, s_2)$.
   It sets $\text{VK} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK}_{\text{one}}\{m^*, \sigma_{\text{one}}\})$ and sends $(\sigma_{\text{one}}, \text{VK})$ to $\mathcal{A}$.

Analysis Let $p_{\mathcal{A},i}$ denote the probability that $\mathcal{A}$ outputs 1 in Hyb_i.

Claim 5.4. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|p_{\mathcal{A},0} - p_{\mathcal{A},1}| \leq \text{negl}(\lambda)$.

Proof. To use the security of $i\mathcal{O}$, we need to argue that $\textsf{Prog-VK}\{K_1, K_2, x\}$ and $\textsf{Prog-VK-Alt}\{m^*, K_1\{m^*\}, K_2, y, \sigma_{\text{one}}\}$ have identical functionality. However, this follows directly from the correctness of F and the fact that PRG is injective. ∎

Claim 5.5. Assuming F is a secure puncturable PRF, for any PPT $\mathcal{A}$, $|p_{\mathcal{A},1} - p_{\mathcal{A},2}| \leq \text{negl}(\lambda)$.

Proof. Now that the program being obfuscated only depends upon the punctured key $K_1\{m^*\}$, we can replace $F(K_1, m^*)$ with a random value by invoking the security of the punctured PRF. ∎

Claim 5.6. Hyb_2 and Hyb_3 are identical, and hence, $p_{\mathcal{A},2} = p_{\mathcal{A},3}$.

Proof. This follows immediately because we have merely made a syntactic change and re-parameterized the same uniform distribution over $\{0,1\}^{\lambda}$. ∎

Claim 5.7. Assuming PRG is a secure pseudorandom generator, for any PPT $\mathcal{A}$, $|p_{\mathcal{A},3} - p_{\mathcal{A},4}| \leq \text{negl}(\lambda)$.

Proof. Now that the preimage x is no longer referenced, we can rely on the security of PRG to change y from being a random image of PRG to a uniformly random string in $\{0,1\}^{2\lambda}$. ∎

Claim 5.8. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT $\mathcal{A}$, $|p_{\mathcal{A},4} - p_{\mathcal{A},5}| \leq \text{negl}(\lambda)$.

Proof. Note that with overwhelming probability, a uniformly random y does not lie in the range of PRG, and hence Step 1(a) of $\textsf{Prog-VK-Alt}$ always outputs 0. Therefore $\textsf{Prog-VK-Alt}$ and $\textsf{Prog-VK}_{\text{one}}$ are functionally identical with all but negligible probability, and in this case the $i\mathcal{O}$ security guarantee applies. ∎

Claim 5.9. Hyb_5 and Hyb_6 are identical, and hence, $p_{\mathcal{A},5} = p_{\mathcal{A},6}$.

Proof. This is simply a reversal of our prior syntactic change. ∎

Claim 5.10. Assuming F is a secure puncturable PRF, for any PPT $\mathcal{A}$, $|p_{\mathcal{A},6} - p_{\mathcal{A},7}| \leq \text{negl}(\lambda)$.

Proof. This is again simply a reversal of our prior change, and follows analogously from the security guarantee of the puncturable PRF. ∎

Combining the above claims, it follows that

$$|\Pr[\mathcal{A} \text{ outputs 1 in } \textsf{Exp-VK}_{\text{one}}|b=0] - \Pr[\mathcal{A} \text{ outputs 1 in } \textsf{Exp-VK}_{\text{one}}|b=0]| \leq \text{negl}(\lambda).$$

∎

Lemma 5.3. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F a secure puncturable PRF and PRG a secure pseudorandom generator, any PPT adversary $\mathcal{A}$ has negligible advantage in $\textsf{Exp-VK}_{\text{abo}}(1^{\lambda}, \mathcal{S}, \mathcal{A})$.

Proof. Let $\mathcal{A}$ be a PPT adversary such that $\textsf{Exp-VK}_{\text{abo}}(1^{\lambda}, \mathcal{S}, \mathcal{A}) = \epsilon$. As in the previous proof, we will define a sequence of hybrids $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_4$, where Hyb_0 and Hyb_4 correspond to the two modes of $\textsf{Exp-VK}_{\text{abo}}(1^{\lambda}, \mathcal{S}, \mathcal{A})$.

Hyb_0 In this experiment, the challenger outputs $(\mathrm{SK}_{\mathrm{abo}}, \mathrm{VK})$ as in the original scheme.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   It computes $\mathrm{SK}_{\mathrm{abo}} \leftarrow i\mathcal{O}(\textsf{Prog-SK}_{\mathrm{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\mathrm{VK} \leftarrow i\mathcal{O}(\textsf{Prog-VK}\{K_1, K_2, x\})$.
   It sends $(\mathrm{SK}_{\mathrm{abo}}, \mathrm{VK})$ to $\mathcal{A}$.

Hyb_1 In this experiment, the challenger uses the program $\textsf{Prog-VK-Alt}'$ (defined in Figure 16) to compute the verification key VK.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   It sets $w = \mathrm{PRG}(F(K_2, m^*))$. It computes $\mathrm{SK}_{\mathrm{abo}} \leftarrow i\mathcal{O}(\textsf{Prog-SK}_{\mathrm{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\mathrm{VK}' \leftarrow i\mathcal{O}(\textsf{Prog-VK-Alt}'\{m^*, K_1, K_2\{m^*\}, w, x\})$.
   It sends $(\mathrm{SK}_{\mathrm{abo}}, \mathrm{VK})$ to $\mathcal{A}$.

$\textsf{Prog-VK-Alt}'$

Constants Message $m^* \in \mathcal{M}$, punctured PRF keys $K_1, K_2\{m^*\} \in \mathcal{K}_p$, $w \in \{0,1\}^{2\lambda}$, $x \in \{0,1\}^\lambda$.
Inputs Message $m \in \mathcal{M}$, signature $\sigma = (\sigma_1, \sigma_2) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$.

1. If $m \neq m^*$
   (a) If $\sigma_1 = F(K_1, m) \oplus x$ **and** $\sigma_2 = F.\mathsf{eval}(K_2\{m^*\}, m)$ output 1. Else output 0.
2. Else,
   (a) If $\sigma_1 = F(K_1, m) \oplus x$ **and** $w = \mathrm{PRG}(\sigma_2)$ output 1. Else output 0.

Figure 16: $\textsf{Prog-VK-Alt}'$

Hyb_2 This hybrid is similar to the previous one, except that the challenger computes $w = \mathrm{PRG}(r)$ for some random $r \leftarrow \{0,1\}^\lambda$.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   it chooses $r \leftarrow \{0,1\}^\lambda$ and sets $w = \mathrm{PRG}(r)$, computes $\mathrm{SK}_{\mathrm{abo}} \leftarrow i\mathcal{O}(\textsf{Prog-SK}_{\mathrm{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\mathrm{VK}' \leftarrow i\mathcal{O}(\textsf{Prog-VK-Alt}'\{m^*, K_1, K_2\{m^*\}, w, x\})$.
   It sends $(\mathrm{SK}_{\mathrm{abo}}, \mathrm{VK})$ to $\mathcal{A}$.

Hyb_3 In this experiment, the challenger chooses a uniformly random 2λ bit for w instead of using PRG to compute it.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   Next, it chooses $w \leftarrow \{0,1\}^{2\lambda}$. It computes $\mathrm{SK}_{\mathrm{abo}} \leftarrow i\mathcal{O}(\textsf{Prog-SK}_{\mathrm{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\mathrm{VK}' \leftarrow i\mathcal{O}(\textsf{Prog-VK-Alt}'\{m^*, K_1, K_2\{m^*\}, w, x\})$.
   It sends $(\mathrm{SK}_{\mathrm{abo}}, \mathrm{VK})$ to $\mathcal{A}$.

Hyb_4 This experiment is similar to the previous one except that the challenger outputs an obfuscation of $\mathsf{Prog\text{-}VK}_{\mathrm{abo}}$ as the verification key.

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$.
2. Challenger chooses puncturable PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*), K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   It computes $\underline{\mathrm{VK}_{\mathrm{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}VK}_{\mathrm{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})}$, $\mathrm{SK}_{\mathrm{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog\text{-}SK}_{\mathrm{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$.
   It sends $(\mathrm{SK}_{\mathrm{abo}}, \mathrm{VK})$ to $\mathcal{A}$.

Analysis Let $p_{i,\mathcal{A}}$ denote the probability that $\mathcal{A}$ outputs 1 in Hyb_i.

Claim 5.11. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|p_{0,\mathcal{A}} - p_{1,\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Note that since PRG is injective and F satisfies the correctness property of puncturable PRFs, circuits $\mathsf{Prog\text{-}VK}\{K_1, K_2, x\}$ and $\mathsf{Prog\text{-}VK\text{-}Alt}\{m^*, K_1, K_2\{m^*\}, x\}$ are functionally identical. As a result, their obfuscations are computationally indistinguishable by $i\mathcal{O}$ security, and hence $|p_{0,\mathcal{A}} - p_{1,\mathcal{A}}| \leq \mathrm{negl}(\lambda)$. ∎

Claim 5.12. Assuming F is a secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|p_{1,\mathcal{A}} - p_{2,\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This follows directly from the selective security definition of puncturable PRFs, as only the punctured key $K_2\{m^*\}$ is reflected in the programs given to $\mathcal{A}$. ∎

Claim 5.13. Assuming PRG is a secure pseudorandom generator, for any PPT adversary $\mathcal{A}$, $|p_{2,\mathcal{A}} - p_{3,\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This follows immediately from the security of PRG, as the preimage r is uniformly random and only w is hardwired into the verification program. ∎

Claim 5.14. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|p_{3,\mathcal{A}} - p_{4,\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Note that since w is chosen uniformly at random, with overwhelming probability, it does not lie in the range of PRG. In this case, $\mathsf{Prog\text{-}VK\text{-}Alt}'\{m^*, K_1, K_2\{m^*\}, w, x\}$ outputs 0 on all inputs with message component m^*. Hence, $\mathsf{Prog\text{-}VK\text{-}Alt}'\{m^*, K_1, K_2\{m^*\}, w, x\}$ and $\mathsf{Prog\text{-}VK}_{\mathrm{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\}$ are functionally identical (with overwhelming probability). ∎

Combining the above claims, it follows that any PPT adversary has negligible advantage in $\mathsf{Exp\text{-}VK}_{\mathrm{abo}}$. ∎

Lemma 5.4. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F a secure puncturable PRF and PRG a secure pseudorandom generator, any PPT adversary $\mathcal{A}$ has negligible advantage in $\mathsf{Exp\text{-}Spl}(1^\lambda, \mathcal{S}, \mathcal{A})$.

Proof. Let $\mathcal{A}$ be a PPT adversary with advantage ϵ in $\mathsf{Exp\text{-}Spl}(1^\lambda, \mathcal{S}, \mathcal{A})$. We will define a sequence of hybrids $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_4$ and compare $\mathcal{A}$'s advantage in each of these hybrids.

Hyb_0 In this experiment, the challenger sends all components corresponding to the same singing key, verification key pair.

1. $\mathcal{A}$ sends message $m^* \in \mathcal{M}$.
2. Challenger chooses PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \in \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   Next, it computes $\sigma_{\text{one}} = (F(K_1, m^*) \oplus x, F(K_2, m^*))$, $\text{VK}_{\text{one}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{one}}\{m^*, \sigma_{\text{one}}\})$, $\text{SK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{SK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\text{VK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$.
   It sends $(\sigma_{\text{one}}, \text{VK}_{\text{one}}, \text{SK}_{\text{abo}}, \text{VK}_{\text{abo}})$ to $\mathcal{A}$.

Hyb_1 In this hybrid, the challenger replaces $F(K_1, m^*)$ and $F(K_2, m^*)$ with uniformly random λ bit strings.

1. $\mathcal{A}$ sends message $m^* \in \mathcal{M}$.
2. Challenger chooses PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \in \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   Next, it chooses $r_1 \leftarrow \{0,1\}^\lambda$, $r_2 \leftarrow \{0,1\}^\lambda$ and sets $\sigma_{\text{one}} = (r_1 \oplus x, r_2)$.
   It computes $\text{VK}_{\text{one}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{one}}\{m^*, \sigma_{\text{one}}\})$, $\text{SK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{SK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\text{VK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and sends $(\sigma_{\text{one}}, \text{VK}_{\text{one}}, \text{SK}_{\text{abo}}, \text{VK}_{\text{abo}})$ to $\mathcal{A}$.

Hyb_2 This experiment is exactly identical to the previous one, except for a notational change, where the first component of σ_{one} is set to be uniformly random.

1. $\mathcal{A}$ sends message $m^* \in \mathcal{M}$.
2. Challenger chooses PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \in \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   Next, it chooses $r_1 \leftarrow \{0,1\}^\lambda$, $r_2 \leftarrow \{0,1\}^\lambda$ and sets $\sigma_{\text{one}} = (r_1, r_2)$.
   It computes $\text{VK}_{\text{one}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{one}}\{m^*, \sigma_{\text{one}}\})$, $\text{SK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{SK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\text{VK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and sends $(\sigma_{\text{one}}, \text{VK}_{\text{one}}, \text{SK}_{\text{abo}}, \text{VK}_{\text{abo}})$ to $\mathcal{A}$.

Hyb_3 In this experiment, the challenger chooses $x' \leftarrow \{0,1\}^\lambda$ and uses x' to compute the first component of σ_{one}. This is a notational change, and this hybrid is identical to the previous one.

1. $\mathcal{A}$ sends message $m^* \in \mathcal{M}$.
2. Challenger chooses PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \in \{0,1\}^\lambda$, $x' \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   Next, it chooses $r_1 \leftarrow \{0,1\}^\lambda$, $r_2 \leftarrow \{0,1\}^\lambda$ and sets $\sigma_{\text{one}} = (r_1 \oplus x', r_2)$.
   It computes $\text{VK}_{\text{one}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{one}}\{m^*, \sigma_{\text{one}}\})$, $\text{SK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{SK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\text{VK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and sends $(\sigma_{\text{one}}, \text{VK}_{\text{one}}, \text{SK}_{\text{abo}}, \text{VK}_{\text{abo}})$ to $\mathcal{A}$.

Hyb_4 In this experiment, the challenger chooses keys K_1', K_2' and computes the signature σ_{one} using K_1', K_2'.

1. $\mathcal{A}$ sends message $m^* \in \mathcal{M}$.
2. Challenger chooses PRF keys $K_1 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2 \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_1' \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_2' \leftarrow F.\mathsf{setup}(1^\lambda)$ and $x \in \{0,1\}^\lambda$, $x' \leftarrow \{0,1\}^\lambda$.
   It computes $K_1\{m^*\} \leftarrow F.\mathsf{puncture}(K_1, m^*)$, $K_2\{m^*\} \leftarrow F.\mathsf{puncture}(K_2, m^*)$.
   Next, it sets $\sigma_{\text{one}}' = (F(K_1', m^*) \oplus x', F(K_2', m^*))$.
   It computes $\text{VK}_{\text{one}}' \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{one}}\{m^*, \sigma_{\text{one}}'\})$, $\text{SK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{SK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and $\text{VK}_{\text{abo}} \leftarrow i\mathcal{O}(\mathsf{Prog}\text{-}\text{VK}_{\text{abo}}\{m^*, K_1\{m^*\}, K_2\{m^*\}, x\})$ and sends $(\sigma_{\text{one}}', \text{VK}_{\text{one}}', \text{SK}_{\text{abo}}, \text{VK}_{\text{abo}})$ to $\mathcal{A}$.

Analysis We will now analyse A's advantage in the above experiments. As before, let $p_{i,\mathcal{A}}$ denote the probability that $\mathcal{A}$ outputs 1 in Hyb_i.

Claim 5.15. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|p_{0,\mathcal{A}} - p_{1,\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. Since only the punctured keys are reflected in the programs given to $\mathcal{A}$, it follows from the security of the puncturable PRF that we can replace $F(K_1, m^*)$ and $F(K_2, m^*)$ with uniformly random strings. ■

Claim 5.16. Experiments Hyb_1 and Hyb_2 are identical, and therefore, $p_{1,\mathcal{A}} = p_{2,\mathcal{A}}$.

Proof. This is just a notation change, as the distribution of σ_{one} is the same. ■

Claim 5.17. Experiments Hyb_2 and Hyb_3 are identical, and therefore, $p_{2,\mathcal{A}} = p_{3,\mathcal{A}}$.

Proof. This is similarly just a notation change that does not affect the distribution of σ_{one}. ■

Claim 5.18. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $|p_{3,\mathcal{A}} - p_{4,\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. Here, we can apply the selective security for the puncturable PRF in reverse, replace random strings with the values $F(K_1', m^*)$ and $F(K_2', m^*)$ at the punctured points. ■

Noting that Hyb_0 and Hyb_4 represent the two scenarios in $\mathsf{Exp\text{-}Spl}(1^\lambda, \mathcal{S}, \mathcal{A})$ and combining the above claims, it follows that any PPT adversary $\mathcal{A}$ has negligible advantage in $\mathsf{Exp\text{-}Spl}(1^\lambda, \mathcal{S}, \mathcal{A})$.

■

6 Message Hiding Encodings

We will now describe a simpler primitive called *message hiding encodings* for Turing machines. Let M be a Turing machine, inp an input to M, and T an integer. Recall that $\Pi_M^T(\mathsf{inp})$ denotes whether M accepts inp within T steps. A message hiding encoding scheme MHE for message space $\mathcal{M}$ consists of algorithms $\mathsf{MHE.enc}$ and $\mathsf{MHE.dec}$ described as follows.

$\mathsf{MHE.enc}(1^\lambda, M, T, \mathsf{inp}, \mathsf{msg})$ The encoding algorithm takes as input the security parameter λ (in unary), the description of a Turing machine M, time bound T (in binary), input inp and message $\mathsf{msg} \in \mathcal{M}$. It outputs an encoding enc.

$\mathsf{MHE.dec}(1^\lambda, M, \mathsf{inp}, T, \mathsf{enc})$ The decoding algorithm takes as input the security parameter λ (in unary), the description of a Turing machine M, time bound T and encoding enc. It outputs either a message $\mathsf{msg} \in \mathcal{M}$ or $\perp$.

Correctness For all Turing machines M, time bounds T, inputs inp and messages $\mathsf{msg} \in \mathcal{M}$, if $\Pi_M^T(\mathsf{inp}) = 1$, then

$$\mathsf{MHE.dec}(1^\lambda, M, \mathsf{inp}, T, \mathsf{MHE.enc}(1^\lambda, M, T, \mathsf{inp}, \mathsf{msg})) = \mathsf{msg}.$$

Efficiency For efficiency, we require that both $\mathsf{MHE.enc}$ should run in time $\mathsf{poly}(\lambda, |\mathsf{M}|, |\mathsf{inp}|, \lg \mathsf{T})$, while $\mathsf{MHE.dec}$'s running time is bounded by $\mathsf{poly}(\lambda, |\mathsf{M}|, |\mathsf{inp}|, \lg \mathsf{T}, \mathsf{t}^*)$, where t^* is the time at which M halts on input inp. Note that this implies the size of encoding enc is bounded by $\mathsf{poly}(\lambda, |\mathsf{M}|, |\mathsf{inp}|, \lg \mathsf{T})$.

Security The security notion for message hiding encoding schemes can be formalized as follows.

Definition 6.1. A message hiding encoding scheme MHE for message space $\mathcal{M}$ is secure if for all security parameters λ, messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}$, Turing machines M, time bounds T, inputs inp such that $\Pi_M^T(\mathsf{inp}) = 0$, for all PPT adversaries $\mathcal{A}$,

$$|\Pr[\mathcal{A}(\mathsf{MHE.enc}(1^\lambda, M, T, \mathsf{inp}, \mathsf{msg}_b)) = b] - 1/2| \leq \mathrm{negl}(\lambda).$$

6.1 Construction

We now describe our message hiding encoding scheme for Turing machines and message space $\mathcal{M}$. We will assume the number of states of any input Turing machine, and the input time bound T are polynomial in λ, and therefore expressible using λ bits. 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 an accumulator for message space Σ_{tape} with accumulated value of size ℓ_{Acc} bits, Itr = ($\mathsf{Setup\text{-}Itr}$, $\mathsf{Setup\text{-}Itr\text{-}Enforce}$, $\mathsf{Iterate}$) an iterator for message space $\{0,1\}^{2\lambda+\ell_{\mathsf{Acc}}}$ with iterated value of size ℓ_{Itr} bits and $\mathcal{S} = (\mathsf{Setup\text{-}Spl}, \mathsf{Sign\text{-}Spl}, \mathsf{Verify\text{-}Spl}, \mathsf{Split}, \mathsf{Sign\text{-}Spl\text{-}abo})$ a splittable signature scheme with message space $\{0,1\}^{\ell_{\mathsf{Itr}}+\ell_{\mathsf{Acc}}+2\lambda}$. We will assume that $\mathsf{Setup\text{-}Spl}$ uses ℓ_{rnd} bits of randomness. Let F a puncturable PRF with key space $\mathcal{K}$, punctured key space $\mathcal{K}_p$, domain $\{0,1\}^\lambda$, range $\{0,1\}^{\ell_{\text{rnd}}}$ and algorithms $F.\mathsf{setup}$, $F.\mathsf{puncture}$, $F.\mathsf{eval}$. The algorithms $\mathsf{MHE.enc}$ and $\mathsf{MHE.dec}$ are defined as follows.

- $\mathsf{MHE.enc}(1^\lambda, M, T, \mathsf{inp}, \mathsf{msg})$ The encoding algorithm first computes $(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. Let $\ell_{\mathsf{inp}} = |\mathsf{inp}|$. It computes $\widetilde{store}_j = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1, \mathsf{inp}_j)$, $aux_j = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1)$, $\widetilde{w}_j = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_{j-1}, \mathsf{inp}_j, j-1, aux_j)$ for $1 \leq j \leq \ell_{\mathsf{inp}}$. Finally, it sets $w_0 = \widetilde{w}_{\ell_{\mathsf{inp}}}$ and $s_0 = \widetilde{store}_{\ell_{\mathsf{inp}}}$.

  Next, it computes $(\mathrm{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{Setup\text{-}Itr}(1^\lambda, T)$, chooses a puncturable PRF key $K_A \leftarrow F.\mathsf{setup}(1^\lambda)$, and computes an obfuscation $P \leftarrow i\mathcal{O}(\mathsf{Prog}\{M, T, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, K_A\})$ where Prog is defined in Figure 17.

  Let $r_A = F(K_A, 0)$, $(\mathrm{SK}_0, \mathrm{VK}_0) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$ and $\sigma_0 = \mathsf{Sign\text{-}Spl}(\mathrm{SK}_0, (v_0, q_0, w_0, 0))$. It outputs $\mathsf{enc} = (P, w_0, v_0, \sigma_0, store_0)$.

Program Prog

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF key $K_A \in \mathcal{K}$.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$.
3. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
4. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
5. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
6. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output msg.
7. Compute $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
8. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
9. Let $F(K_A, t) = r'_A$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$.
10. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\text{out}})$.
11. Output $\mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 17: Program Prog

- $\mathsf{MHE.dec}(1^\lambda, M, \mathsf{inp}, 1^T, \mathsf{enc})$ Let $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$ and $\mathsf{enc} = (P, w_0, v_0, \sigma_0, store_0)$. Let $\mathsf{pos}_0 = 0$, $\mathsf{st}_0 = q_0$.

  For $i = 1$ to T, compute

  1. Let $(\mathsf{sym}_{i-1}, \pi_{i-1}) = \mathsf{Prep\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, store_{i-1}, \mathsf{pos}_{i-1})$.
  2. Compute $aux_{i-1} \leftarrow \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, store_{-1}, \mathsf{pos}_{i-1})$.
  3. Let $\mathrm{out} = P(i, \mathsf{sym}_{i-1}, \mathsf{pos}_{i-1}, \mathsf{st}_{i-1}, w_{i-1}, v_{i-1}, \pi_{i-1}, aux_{i-1})$. If $\mathrm{out} \in \mathcal{M} \cup \{\perp\}$ output out. Else parse out as $(\mathsf{sym}_{w,i}, \mathsf{pos}_i, \mathsf{st}_i, w_i, v_i, \sigma_i)$.
  4. Compute $store_i = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, store_{i-1}, \mathsf{pos}_{i-1}, \mathsf{sym}_{w,i})$.

The basic idea of this construction is that input can accumulated as a preprocessing step, and then an initial signature is produced. This then allows for the main program to step through the computation, taking one transition of the Turing Machine at a time. We note that the decryption only takes t^* steps rather than T, as we have a condition in line 3. above that breaks out of the for loop when the computation is finished.

Remark 6.1. All obfuscated programs are padded appropriately to be of the same size as the largest program in the corresponding proof hybrids.

Remark 6.2. Note that Prog could receive 'ϵ' as the symbol input. We will assume Prog interprets 'ϵ' as '␣'.

6.2 Proof of Security

Theorem 6.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, MHE is a secure message hiding encoder.

Proof. Suppose there exists a security parameter λ, machine M, time bound T, input inp, messages $\mathsf{msg}_0, \mathsf{msg}_1$ and a PPT adversary $\mathcal{A}$ such that $\Pr[\mathcal{A}(\mathsf{MHE.enc}(1^\lambda, M, T, \mathsf{inp}, \mathsf{msg}_b)) = 1] - 1/2 = \epsilon$ which is non-negligible. To arrive at a contradiction, we will first define a sequence of *outer hybrid* experiments $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_4$ and then show that any two consecutive hybrid experiments are computationally indistinguishable. Let us assume M accepts/rejects inp at time $t^* < T$.

Hyb_0 This hybrid corresponds to the real security game.

Hyb_1 This hybrid is similar to the previous one, except that the challenger chooses another PRF key K_B to be used for signing/verifying 'B' type signature. The challenger computes $K_B \leftarrow F.\mathsf{setup}(1^\lambda)$ and outputs an obfuscation of $\mathsf{Prog\text{-}1}\{M, T, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, K_A, K_B\}$. (defined in Figure 18). In this program, the program checks for 'B' type signatures, and if the incoming message has a 'B' type signature, then the program signs the outgoing message as a 'B' type signature. Also, if the incoming signature is 'B' type and $t < t^*$, then the program rejects if state is q_{rej} or q_{acc}.

We will now define $2t^*$ hybrids $\mathsf{Hyb}_{2,i}$ and $\mathsf{Hyb}'_{2,i}$ for $0 \le i < t^*$ as follows.

$\mathsf{Hyb}_{2,i}$ 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. $(\mathsf{sym}_j, \pi_j) = \mathsf{Prep\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, store_{j-1}, \mathsf{pos}_{j-1})$.
2. $aux_j = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, store_{j-1}, \mathsf{pos}_{j-1})$.
3. $(\mathsf{st}_j, \mathsf{sym}_{w,j}, \beta) = \delta(\mathsf{st}_{j-1}, \mathsf{sym}_{j-1})$.

Prog-1

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $t > t^*$ output $\perp$.
6. If $\alpha \neq$ 'A' **and** $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$'B', output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$'A', output msg.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
14. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 18: Prog-1

4. $w_j = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{j-1}, \mathsf{sym}_{w,j}, \mathsf{pos}_{j-1}, aux_j)$.
5. $v_j = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{j-1}, (\mathsf{st}_{j-1}, w_{j-1}, \mathsf{pos}_{j-1}))$.
6. $store_j = \mathsf{Write\text{-}Store}(\text{PP}_{\mathsf{Acc}}, store_{j-1}, \mathsf{pos}_{j-1}, \mathsf{sym}_{w,j})$.
7. $\mathsf{pos}_j = \mathsf{pos}_{j-1} + \beta$.

It sets $m_i = (v_i, \mathsf{st}_i, w_i, \mathsf{pos}_i)$ and computes the obfuscation of $\mathsf{Prog\text{-}2\text{-}}i\{i, M, T, \mathsf{msg}_b, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, K_A, K_B, m_i\}$ (defined in Figure 19), which accepts only 'A' type signatures for the first i time steps. Also, if the state output is q_{acc} in the first i steps, it outputs $\perp$ instead of outputting message. For the output, if $t = i$ and the message to be signed is the 'correct message', it outputs an 'A' type signature, else it outputs a 'B' type signature. For $t > i$, the type of signature output is the same as type of signature input.

$\mathsf{Hyb}'_{2,i}$ In this hybrid, the challenger outputs the obfuscation of $\mathsf{Prog}'\text{-}2\text{-}i\{i,\ M,\ T,\ \mathsf{msg}_b,\ \text{PP}_{\mathsf{Acc}},\ \text{PP}_{\mathsf{Itr}},\ K_A,\ K_B,\ m_i\}$ (defined in Figure 20), which accepts only 'A' type signatures for the first $i+1$ time steps. If the state is q_{acc} at $(i+1)^{th}$ step, it outputs $\perp$. For the output, if $t = i+1$ and the input message is the 'correct message', it outputs an 'A' type signature, else it outputs a 'B' type signature. For $t > i+1$, the type of signature output is the same as type of signature input.

Hyb_3 In this hybrid, the challenger outputs an obfuscation of $P_3 = \mathsf{Prog\text{-}3}\{M, T, t^*, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, \mathsf{msg}_b, K_A, K_B\}$ (described in Figure 21). Note that $F(K_A, t^*)$ will not be computed in this hybrid. Also, the only inputs for which a 'B' type signature can possibly be output correspond to $t = t^*$.

Hyb_4 In this hybrid, the challenger outputs the obfuscation of $P_b = \mathsf{Prog\text{-}4}\{M, T, t^*, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, K_A, K_B\}$ (defined in Figure 22), a program that outputs $\perp$ for all $t > t^*$, including the case when the signature is a valid 'A' type signature. As a result, it doesn't output msg at any instant, and hence P_0 and P_1 are identical.

Prog-2-i

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$, message m_i.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. Let $r_A = F(K_A, t-1), r_B = F(K_B, t-1)$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\text{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** ($t > t^*$ **or** $\underline{t \leq i}$) output $\perp$.
6. If $\alpha \neq$ 'A' **and** $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$'B', output $\perp$.
    <u>Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$ 'A' **and** $t \leq i$, output $\perp$.</u>
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$, output msg.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
12. Compute $v_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\text{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
14. Let $m_{\text{out}} = (v_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    <u>If $t = i$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$</u> .
    <u>Else if $t = i$ **and** $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.</u>
    <u>Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.</u>
15. Output $\mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\text{out}}, \sigma_{\text{out}}$.

Figure 19: Prog-2-i

Analysis Let $\mathsf{Adv}^x_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in Hyb_x, and $\mathsf{Adv}'^x_{\mathcal{A}}$ the advantage of $\mathcal{A}$ in Hyb'_x.

Lemma 6.1. 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 5.1, 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 contained in Appendix A.1.

Claim 6.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscation, for any PPT $\mathcal{A}$, $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^{2,0}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. Programs Prog-2-0 and Prog-1 are functionally identical. As a result, their obfuscations are computationally indistinguishable, by the security requirement of $i\mathcal{O}$. ∎

Lemma 6.2. Let $1 \leq i \leq t^*$. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF and $\mathcal{S}$ is a splittable signature scheme satisfying definitions 5.1, 5.2, 5.3 and 5.4, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{2,i}_{\mathcal{A}} - \mathsf{Adv}'^{2,i}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

The proof of this lemma is contained in Appendix A.2.

Lemma 6.3. Let $1 \leq i \leq t^*$. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, Itr is an iterator satisfying indistinguishability of Setup (Definition 3.1) and is enforcing (Definition 3.2), and Acc is an accumulator satisfying indistinguishability of Read/Write Setup (Definitions 4.1 and 4.2) and is Read/Write enforcing (Definitions 4.3 and 4.4), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}'^{2,i}_{\mathcal{A}} - \mathsf{Adv}^{2,i+1}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

$\mathsf{Prog}'\text{-}2\text{-}i$

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time t^*, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$, message m_i.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\text{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $(t > t^*$ **or** $t \leq i+1)$ output $\perp$.
6. If $\alpha \neq$ 'A' **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$'B', output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$ 'A' **and** $t \leq i+1$ output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output msg.
11. Compute $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
12. Compute $v_{\text{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\text{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
14. Let $m_{\text{out}} = (v_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = i+1$ **and** $m_{\text{in}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\text{out}})$.
    Else if $t = i+1$ **and** $m_{\text{in}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\text{out}}, \sigma_{\text{out}}$.

Figure 20: $\mathsf{Prog}'\text{-}2\text{-}i$

The proof of this lemma is contained in Appendix A.3.

Lemma 6.4. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_{\mathcal{A}}^{\prime 2,t^*-1} - \mathsf{Adv}_{\mathcal{A}}^{3}| \leq \text{negl}(\lambda)$.

The proof of this lemma is contained in Appendix A.4.

Lemma 6.5. Assuming $\mathcal{S}$ satisfies VK_{rej} indistinguishability (Definition 5.1), $i\mathcal{O}$ is a secure indistinguishability obfuscator and F is a selectively secure pseudorandom function, for any adversary $\mathcal{A}$, $|\mathsf{Adv}_{\mathcal{A}}^{3} - \mathsf{Adv}_{\mathcal{A}}^{4}| \leq \text{negl}(\lambda)$.

The proof of this lemma is contained in Appendix A.5.

Claim 6.2. Any adversary $\mathcal{A}$ has 0 advantage in Hyb_4.

Proof. $\mathsf{Prog}\text{-}4\{M, T, t^*, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, K_A, K_B\}$ is independent of message msg_b. As a result, any adversary has 0 advantage in guessing b. ∎

∎

Prog-3

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time t^*, message msg, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
5. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
6. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
7. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output msg.
8. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
9. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
10. Let $r'_B = F(K_B, t)$, compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
11. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = t^*$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else let $r'_A = F(K_A, t)$,
    compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$,
    $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
12. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 21: Prog-3

Prog-4

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time t^*, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$ output $\perp$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
3. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
4. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
5. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
6. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
7. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
8. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output $\perp$.
9. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
10. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
11. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
12. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = t^*$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
13. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 22: Prog-4

7 Machine Hiding Encodings

In this section, we will describe *machine hiding encodings*. Let M be a Turing machine, x an input to the Turing machine, and T the time bound. As before, let $\Pi_M^T(x)$ be a function runs M on input x for at most

T steps, and if M does not halt in T steps, it outputs 0. A machine hiding encoding scheme McHE consists of algorithms $\mathsf{Mc.enc}$ and $\mathsf{Mc.dec}$ described below.

$\mathsf{Mc.enc}(1^\lambda, M, T, x)$ The encoding algorithm is a randomized algorithm that takes as input the security parameter λ (in unary), the description of a Turing machine M, time bound T (in binary) and an input x. It outputs an encoding enc.

$\mathsf{Mc.dec}(1^\lambda, M, T, x, \mathsf{enc})$ The decoding algorithm takes as input the security parameter λ (in unary), the description of Turing machine M, time bound T (in binary), input x and an encoding enc. It outputs 0/1.

Correctness Fix any security parameter λ, Turing machine M, input x and time bound T. Then, the following holds:

$$\mathsf{Mc.dec}(1^\lambda, M, x, \mathsf{Mc.enc}(1^\lambda, M, T, x)) = \Pi_M^T(x).$$

Efficiency We require that $\mathsf{Mc.enc}$ outputs the encoding in time $\mathsf{poly}(\lambda, |\mathsf{M}|, |\mathsf{x}|, \lg \mathsf{T})$ (which implies the size of encoding is $\mathsf{poly}(\lambda, |\mathsf{M}|, |\mathsf{x}|, \lg \mathsf{T})$), while $\mathsf{Mc.dec}$ runs in time $\mathsf{poly}(\lambda, |\mathsf{M}|, |\mathsf{x}|, \mathsf{t}^*)$, where t^* is the running time of M on input x.

Security We consider the following indistinguishability based security notion. Let M_0, M_1 be any Turing machines, T a time bound and x an input to the Turing machines. M_0 and M_1 are said to be conforming on input x if $M_0(x) = M_1(x)$ and both halt in $t^* < T$ steps.

Definition 7.1. A machine hiding encoding scheme McHE is said to be secure if for all PPT adversaries $\mathcal{A}$, for all security parameters λ, machines M_0, M_1 with equally-sized descriptions, time bounds T and inputs x such that M_0 and M_1 are conforming on input x,

$$|\Pr[\mathcal{A}(\mathsf{Mc.enc}(1^\lambda, M_b, T, x)) = b] - 1/2| \leq \text{negl}(\lambda).$$

Here, when we say that M_0 and M_1 are "conforming" on x, we mean that $M_0(x) = M_1(x)$, and both halt at the same time t^*.

Remark 7.1. For our construction, we will assume the encoding function also receives as input the tape movement function for machine M on input x, and that M_0 and M_1 have identical tape movement functions on input x; that is $\mathsf{tmf}_{M_0,x}$ and $\mathsf{tmf}_{M_1,x}$ are identical functions. We can make this assumption without loss of generality because both M_0 and M_1 can be compiled into oblivious Turing machines OTM_0 and OTM_1 such that they have identical tape movement functionality. The overhead associated with converting them into oblivious Turing machines is only $\mathsf{polylog}(T)$ [PF79].

7.1 Construction

In this section, we will construct a machine hiding encoding scheme $\mathsf{McHE} = (\mathsf{Mc.enc}, \mathsf{Mc.dec})$. This construction is similar to the message hiding encoding scheme construction described in Section 6.1. As mentioned in Remark 7.1, we will assume the encoding function is also given the tape movement function $\mathsf{tmf}_M(\cdot)$.

Let $\mathcal{PKE} = (\mathsf{Setup\text{-}PKE}, \mathsf{Enc\text{-}PKE}, \mathsf{Dec\text{-}PKE})$ be a public key encryption scheme. We will assume $\mathsf{Setup\text{-}PKE}$ uses $\ell_1 = \ell_1(\lambda)$ bits of randomness, and $\mathsf{Enc\text{-}PKE}$ uses $\ell_2 = \ell_2(\lambda)$ bits of randomness, where ℓ_1 and ℓ_2 are polynomials and let $\ell_{\text{rnd}} = \ell_1 + 2\ell_2$. We will let ℓ_3 denote the bit length of ciphertexts produced by $\mathsf{Enc\text{-}PKE}$. Let $i\mathcal{O}$ be a secure indistinguishability obfuscator, 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) a positional accumulator scheme with message space $\{0,1\}^{\ell_3 + \lg T}$ and producing accumulator values of bit length ℓ_{Acc}, $\mathsf{Itr} = (\mathsf{Setup\text{-}Itr}$, $\mathsf{Setup\text{-}Itr\text{-}Enforce}$, $\mathsf{Iterate})$ an iterator for message space $\{0,1\}^{\ell_3 + \ell_{\mathsf{Acc}} + \lg T}$ with iterated value of size ℓ_{Itr} bits and $\mathcal{S} = (\mathsf{Setup\text{-}Spl}, \mathsf{Sign\text{-}Spl}, \mathsf{Verify\text{-}Spl}, \mathsf{Split}, \mathsf{Sign\text{-}Spl\text{-}abo})$ a splittable signature scheme with message space $\{0,1\}^{\ell_{\mathsf{Itr}} + \ell_3 + \ell_{\mathsf{Acc}} + \lg T}$. For simplicity of notation, we will assume $\mathsf{Setup\text{-}Spl}$ uses $\ell_{\text{rnd}}(\lambda)$ bits of randomness.

Let F a puncturable PRF with key space $\mathcal{K}$, punctured key space $\mathcal{K}_p$, domain $[T]$, range $\{0,1\}^{\ell_{\mathrm{rnd}}(\lambda)}$ and algorithms $F.\mathsf{setup}$, $F.\mathsf{puncture}$, $F.\mathsf{eval}$. The algorithms $\mathsf{Mc.enc}$ and $\mathsf{Mc.dec}$ are defined as follows.

- $\mathsf{Mc.enc}(1^\lambda, M, T, x, \mathsf{tmf})$ The encoding algorithm first chooses puncturable PRF keys $K_E \leftarrow F.\mathsf{setup}(1^\lambda)$, $K_A \leftarrow F.\mathsf{setup}(1^\lambda)$. K_E will be used for computing an encryption of the symbol and state, and K_A to compute the secret key/verification key for signature scheme. Let $(r_{0,1}, r_{0,2}, r_{0,3}) = F(K_E, 0)$, $(\mathsf{pk}, \mathsf{sk}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{0,1})$.

  It computes $(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. Let $\ell_{\mathsf{inp}} = |x|$. It encrypts each bit of x separately; that is, it computes $\mathsf{ct}_i = \mathsf{Enc\text{-}PKE}(\mathsf{pk}, x_i)$ for $1 \leq i \leq \ell_{\mathsf{inp}}$. These ciphertexts are 'accumulated' using the accumulator. It computes $\widetilde{store}_j = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1, (\mathsf{ct}_j, 0))$, $aux_j = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1)$, $\widetilde{w}_j = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_{j-1}, \mathsf{inp}_j, j-1, aux_j)$ for $1 \leq j \leq \ell_{\mathsf{inp}}$. Finally, it sets $w_0 = \widetilde{w}_{\ell_{\mathsf{inp}}}$ and $s_0 = \widetilde{store}_{\ell_{\mathsf{inp}}}$.

  Next, it computes $(\mathrm{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{Setup\text{-}Itr}(1^\lambda, T)$, Finally, it computes an obfuscation $P \leftarrow i\mathcal{O}(\mathsf{Prog}\{M,$ T, $\mathrm{PP}_{\mathsf{Acc}}$, $\mathrm{PP}_{\mathsf{Itr}}$, K_E, $K_A\})$ where Prog is defined in Figure 23.

  It computes $\mathsf{ct}_{\mathsf{st}} \leftarrow \mathsf{Enc\text{-}PKE}(\mathsf{pk}, q_0)$. Let $r_A = F(K_A, 0)$, $(\mathrm{SK}_0, \mathrm{VK}_0) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$ and $\sigma_0 = \mathsf{Sign\text{-}Spl}(\mathrm{SK}_A, (v_0, \mathsf{ct}_{\mathsf{st}}, w_0, 0))$. It outputs $\mathsf{enc} = (P, w_0, v_0, \sigma_0, store_0)$.

Program Prog

Constants: Turing machine $M = \langle Q, \Sigma_{\mathrm{tape}}, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}\rangle$, time bound T, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A \in \mathcal{K}$.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\mathrm{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\mathrm{in}}$, accumulator value $w_{\mathrm{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\mathrm{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\mathrm{out}} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, (\mathsf{ct}_{\mathsf{sym},\mathrm{in}}, \mathsf{lw}), \mathsf{pos}_{\mathrm{in}}, \pi) = 0$ or $\mathsf{lw} \geq t$, output $\perp$.
3. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
4. Let $m_{\mathrm{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 0$ output $\perp$.
5. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\mathrm{in}})$.
6. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\mathrm{in}})$.
7. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
8. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{rej}}$ output 0.
9. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ output 1.
10. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\mathrm{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\mathrm{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
11. Compute $w_{\mathrm{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, (\mathsf{ct}_{\mathsf{sym},\mathrm{out}}, t), \mathsf{pos}_{\mathrm{in}}, aux)$. If $w_{\mathrm{out}} = Reject$, output $\perp$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}))$.
13. Let $F(K_A, t) = r'_{S,A}$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
14. Let $m_{\mathrm{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\mathrm{out}}, w_{\mathrm{out}}, \mathsf{pos}_{\mathrm{out}})$ and $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\mathrm{out}})$.
15. Output $\mathsf{pos}_{\mathrm{in}}, \mathsf{ct}_{\mathsf{sym},\mathrm{out}}, \mathsf{ct}_{\mathsf{ct},\mathrm{out}}, w_{\mathrm{out}}, v_{\mathsf{out}}, \sigma_{\mathrm{out}}$.

Figure 23: Program Prog

- $\mathsf{Mc.dec}(\mathsf{enc})$ The decoding algorithm receives as input $\mathsf{enc} = ((P, \mathsf{ct}_{\mathsf{st},0}, w_0, v_0, \sigma_0, store_0))$. Let $\mathsf{pos}_0 = 0$. For $i = 1$ to T,

  1. Let $((\mathsf{ct}_{\mathsf{sym},\mathsf{lw}}, \mathsf{lw}), \pi) = \mathsf{Prep\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, store_{i-1}, \mathsf{pos}_{i-1})$.
  2. Let $aux = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, store_{i-1}, \mathsf{pos}_{i-1})$.
  3. Compute $(\mathsf{pos}_i, (\mathsf{ct}_{\mathsf{sym},i}, \mathsf{lw}), \mathsf{ct}_{\mathsf{st},i}, w_i, v_i, \sigma_i) = P(t, (\mathsf{ct}_{\mathsf{sym},\mathsf{lw}}, \mathsf{lw}), \mathsf{ct}_{\mathsf{st},i-1}, w_{i-1}, v_{i-1}, \sigma_{i-1}, aux, \pi)$. If P has output 0, 1, or $\perp$, then output the same.
  4. Otherwise, compute $store_i = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, store_{i-1}, \mathsf{pos}_i, (\mathsf{ct}_{\mathsf{sym},i}, i))$.

7.2 Proof of Security

Theorem 7.1. Assuming $\mathcal{PKE}$ is IND-CPA secure, $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, McHE is a secure machine hiding encoding scheme (Definition 7.1).

Proof. Consider any Turing machines M_0 and M_1, input x and time bound T such that M_0 and M_1 are conforming on input x. Let $\mathsf{tmf}(\cdot)$ be the tape movement function corresponding to M_0 and M_1, and let $t^* < T$ be the instance at which the machines halt on input x. For the proof, we wil define a sequence of 'outer' hybrid experiments $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}$, and then show that any two outer hybrids are computationally indistinguishable.

Hyb_0 This hybrid corresponds to the real security game. The challenger chooses $b \leftarrow \{0,1\}$ and honestly computes $\mathsf{enc} \leftarrow \mathsf{Mc.enc}(1^\lambda, M_b, T, x, \mathsf{tmf}(\cdot))$. It sends enc to $\mathcal{A}$ and $\mathcal{A}$ sends its guess b'.

Hyb_1 In this hybrid, the challenger outputs an obfuscation of $\mathsf{Prog}\text{-}1\{t^*, K_A, K_E, b^*\}$ (defined in Figure 24) as part of the encoding. This program is similar to Prog, however, for input $t > t^*$, it outputs $\perp$. At $t = t^*$, it outputs b^*, which is hardwired by the challenger to be $M_b(x)$.

Prog-1

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \leq T$, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A \in \mathcal{K}$, output b^*.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ or $\mathsf{lw} \geq t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
6. If $t = t^*$, output b^*.
7. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
8. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
9. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
11. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
12. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
13. Compute $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
14. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
15. Let $F(K_A, t) = r'_{S,A}$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
16. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\text{out}})$.
17. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 24: Prog-1

Next, we define a sequence of hybrids $\mathsf{Hyb}_{2,i}$ and $\mathsf{Hyb}'_{2,i}$, where $1 \leq i \leq t^*$. Let erase be a symbol not present in Σ_{tape}.

$\mathsf{Hyb}_{2,i}$ In this hybrid, the challenger outputs an obfuscation of $\mathsf{Prog}\text{-}2\text{-}i\{i, t^*, K_E, K_A, b^*\}$ as part of the encoding. $\mathsf{Prog}\text{-}2\text{-}i$ also rejects on input $t > t^*$, and outputs b^* on t^* if the signature is the correct one. For

$t < i$, its input output behavior is similar to that of Prog. However, for $i \le t < t^*$, on receiving a valid signature, it simply outputs encryptions of erase as the encryption of the state and symbol. It accumulates and iterates accordingly.

Prog-2-i

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \le T$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_A \in \mathcal{K}$, output b^*.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ or $\mathsf{lw} \ge t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
7. If $t = t^*$, output b^*.
8. If $i \le t < t^*$
   (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$. $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
9. Else
   (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
   (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
   (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
   (d) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
   (e) If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
   (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
10. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
11. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
12. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
13. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 25: Prog-2-i

$\mathsf{Hyb}'_{2,i}$ In this hybrid, the challenger chooses $b \in \{0,1\}$, runs M_b for $i-1$ steps and computes the state st^* and symbol sym^* written at $(i-1)^{th}$ step. Next, it computes $(r_{i-1,1}, r_{i-1,2}, r_{i-1,3}) = F(K_E, i-1)$, $(\mathsf{pk}, \mathsf{sk}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{i-1,1})$, $\mathsf{ct}_1 = \mathsf{Enc\text{-}PKE}(\mathsf{pk}, \mathsf{sym}^*; r_{i-1,2})$ and $\mathsf{ct}_2 = \mathsf{Enc\text{-}PKE}(\mathsf{pk}, \mathsf{st}^*; r_{i-1,3})$. It then computes the obfuscation of $W_{i,b} = \mathsf{Prog}'\text{-}2\text{-}i\{i, t^*, K_E, K_A, \mathsf{ct}_1, \mathsf{ct}_2, b^*\}$, which has the ciphertexts ct_1 and ct_2 hardwired. On input corresponding to step $i-1$, $W_{i,b}$ checks if the signature is valid, and if so, it outputs ct_1 and ct_2 without decrypting.

Analysis Let $\mathsf{Adv}^x_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in hybrid Hyb_x, and $\mathsf{Adv}'^x_{\mathcal{A}}$ the advantage of $\mathcal{A}$ in Hyb'_x.

Lemma 7.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}} \le \text{negl}(\lambda)$.

Prog$'$-2-i

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_{\mathsf{st}} \in \mathcal{K}$, output b^*, ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$.
Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ or $\mathsf{lw} \geq t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
7. If $t = t^*$, output b^*.
8. If $i \leq t < t^*$
   (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
9. Else if $t = i - 1$,
   (a) Set $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{ct}_1$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{ct}_2$.
10. Else
   (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
   (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
   (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
   (d) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
   (e) If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
   (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
14. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 26: Prog$'$-2-i

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

Claim 7.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any adversary $\mathcal{A}$, $\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^{2,t^*}_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. We note that the programs $\mathsf{Prog\text{-}1}$ and $\mathsf{Prog\text{-}2\text{-}}t^*$ are functionally identical. ■

Lemma 7.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{2,i}_{\mathcal{A}} - \mathsf{Adv}'^{2,i}_{\mathcal{A}} \leq \text{negl}(\lambda)$.

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

Lemma 7.3. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, $\mathcal{PKE}$ is $\mathsf{IND\text{-}CPA}$ secure, for any adversary $\mathcal{A}$, $\mathsf{Adv}'^{2,i}_{\mathcal{A}} - \mathsf{Adv}^{2,i-1}_{\mathcal{A}} \leq \text{negl}(\lambda)$.

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

To conclude, we note that $\mathsf{Adv}^{2,1}_{\mathcal{A}} = 0$, as the differences between M_0 and M_1 have all been "erased." ■

7.3 Extensions and Variations

SInce our proof is aimed at achieving machine hiding, it does not erase the input. However, the proof could be easily extended at the end to do this via a simple IND-CPA based hybrids. Note that at the end of our current proofs the no steps actually decrypt the cipertexts. If we add this step of hiding the input, the construction can serve as a secure randomized encoding.

Acknowledgements

We very grateful to Shafi Goldwasser for enlightening discussions on applications of message hiding encodings at an early stage of this project. We are also grateful to Yuval Ishai and Amit Sahai for helpful discussions and comments. Finally, we thank Yuval Ishai for providing us a copy of [IW14] at no cost.

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. `http://eprint.iacr.org/`.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, pages 119–135, 2001.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120, 2013.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 52–73, 2014.

[BdM93] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, pages 274–285, 1993.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

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

[BGT14] Nir Bitansky, Sanjam Garg, and Sidharth Telang. Succinct randomized encodings and their applications. Cryptology ePrint Archive, Report 2014/771, 2014. `http://eprint.iacr.org/`.

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

[CHJV14] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and ram programs. Cryptology ePrint Archive, Report 2014/769, 2014. `http://eprint.iacr.org/`.

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

[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.

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 151–160, 1998.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 536–553, 2013.

[GLSW14] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014. `http://eprint.iacr.org/`.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 426–443, 2014.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 294–304, 2000.

[IPS14] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-Coin Differing-Inputs Obfuscation and Its Applications. Manuscript, 2014.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 650–662, 2014.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM Conference on Computer and Communications Security*, pages 669–684, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 485–494, 2014.

[LP14] Huijia Lin and Rafael Pass. Succinct garbling schemes and applications. Cryptology ePrint Archive, Report 2014/766, 2014. `http://eprint.iacr.org/`.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

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

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91, 1982.

A Proofs for Section 6

A.1 Proof of Lemma 6.1

Proof Intuition Let us consider the differences between Prog and Prog-1.

1. For inputs corresponding to $t > t^*$, both programs are identical.
2. For inputs corresponding to $t \leq t^*$, Prog-1 first checks if it is an 'A' type signature. If not, it checks if it a 'B' type signature. If the incoming signature is a 'B' type signature, then the program cannot output msg; instead, it aborts if $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$. However, if $\mathsf{st}_{\mathrm{out}}$ is neither q_{acc} nor q_{rej}, then the output signature is of the same type as the input one.

So, we need to allow 'B' type signatures for steps $1 \leq t \leq t^*$. We do this in a 'top-down' manner, and define intermediate hybrid experiments $H_{t^*}, \ldots, H_0$, where H_i outputs Prog-0-i, which allows only 'A' type signatures for $t \leq i$ or $t > t^*$.

First, note that Prog-0-i can be modified into another functionally equivalent program which uses a 'reject' verification key if $t = i$ and 'A' verification fails. While using this reject verification key, we can add additional logic to the program, ensuring that it rejects if it verifies a 'B' type signature at $t = i$ and it reaches q_{acc} (the program never reaches this piece of code and hence functionality is preserved). Finally, we observe that Prog-0-i does not require $F(K_B, i-1)$, so K_B can be punctured at $i-1$; ensuring that the adversary has no information about the secret key derived from $F(K_B, i-1)$. Using $\mathrm{VK}_{\mathrm{rej}}$ indistinguishability, we can replace the reject verification key with a valid verification key.

Formal proof We will first define t^* intermediate hybrids $H_0, \ldots, H_{t^*}$, and then show that any two consecutive hybrids are computationally indistinguishable.

Hybrid H_i In this experiment, the challenger outputs an obfuscation of Prog-0-$i\{i, M, T, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, K_A, K_B\}$ (defined in Figure 27).

Clearly, H_{t^*} corresponds to Hyb_0 and H_0 to Hyb_1. Therefore, it suffices to show that H_i and H_{i-1} are computationally indistinguishable. Let $\mathsf{Adv}^i_{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in H_i.

Claim A.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 5.1, 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 $H_{i,a}, \ldots, H_{i,f}$.

Hybrid $H_{i,a}$ In this hybrid, the challenger outputs an obfuscation of Prog-0-i-$a\{i, M, T, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, K_A, K_B\}$ (described in Figure 28), which is functionally identical to Prog-0-$i\{i, M, T, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, K_A, K_B\}$.

Hybrid $H_{i,b}$ In this hybrid, the challenger first punctures the PRF key K_B on input $i-1$. It computes $K_B\{i-1\} \leftarrow F.\mathsf{puncture}(K_B, i-1)$. Next, it computes $r_C = F(K_B, i-1)$ and $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$. It hardwires $K_B\{i-1\}$ and $\mathrm{VK}_{C,\mathrm{rej}}$ in the program Prog-0-i-$b\{i, M, T, \mathrm{PP}_{\mathsf{Acc}}, \mathrm{PP}_{\mathsf{Itr}}, K_A, K_B\{i-1\}, \mathrm{VK}_{C,\mathrm{rej}}\}$ (defined in Figure 29).

Prog-0-i

Constants: i, Turing machine $M = \langle Q, \Sigma_{\mathrm{tape}}, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\mathrm{in}} \in \Sigma_{\mathrm{tape}}$, position $\mathsf{pos}_{\mathrm{in}} \in [T]$, state $\mathsf{st}_{\mathrm{in}} \in Q$, accumulator value $w_{\mathrm{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, \mathsf{sym}_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $\alpha =$‘-’ and $m_{\mathrm{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$ set $\alpha =$‘A’ .
5. If $\alpha =$‘-’ **and** $(t > t^*$ **or** $\underline{t \leq i})$ output $\perp$.
6. If $\alpha =$ ‘-’ **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$, set $\alpha =$‘B’.
7. If $\alpha =$‘-’, output $\perp$.
8. Let $(\mathsf{st}_{\mathrm{out}}, \mathsf{sym}_{\mathrm{out}}, \beta) = \delta(\mathsf{st}_{\mathrm{in}}, \mathsf{sym}_{\mathrm{in}})$ and $\mathsf{pos}_{\mathrm{out}} = \mathsf{pos}_{\mathrm{in}} + \beta$.
9. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ **and** $\alpha =$‘B’, output $\perp$.
    Else if $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ and $\alpha =$‘A’, output msg.
11. Compute $w_{\mathrm{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, \mathsf{sym}_{\mathrm{out}}, \mathsf{pos}_{\mathrm{in}}, aux)$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}))$.
13. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
14. Let $m_{\mathrm{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\mathrm{out}}, w_{\mathrm{out}}, \mathsf{pos}_{\mathrm{out}})$ and $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\mathrm{out}})$.
15. Output $\mathsf{pos}_{\mathrm{out}}, \mathsf{sym}_{\mathrm{out}}, \mathsf{st}_{\mathrm{out}}, w_{\mathrm{out}}, v_{\mathsf{out}}, \sigma_{\mathrm{out}}$.

Figure 27: Prog-0-i

Prog-0-i-a

Constants: i, Turing machine $M = \langle Q, \Sigma_{\mathrm{tape}}, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\mathrm{in}} \in \Sigma_{\mathrm{tape}}$, position $\mathsf{pos}_{\mathrm{in}} \in [T]$, state $\mathsf{st}_{\mathrm{in}} \in Q$, accumulator value $w_{\mathrm{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, \mathsf{sym}_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$, $\underline{\mathrm{VK} = \mathrm{VK}_{B,\mathrm{rej}}}$.
3. Let $\alpha =$‘-’ and $m_{\mathrm{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$ set $\alpha =$‘A’ .
5. If $\alpha =$‘-’ **and** $(t > t^*$ **or** $\underline{t \leq i-1})$ output $\perp$.
6. <u>If $\alpha =$‘-’ **and** $t = i$ **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 0$ output $\perp$.</u>
7. If $\alpha =$ ‘-’ **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$, set $\alpha =$‘B’.
8. If $\alpha =$‘-’, output $\perp$.
9. Let $(\mathsf{st}_{\mathrm{out}}, \mathsf{sym}_{\mathrm{out}}, \beta) = \delta(\mathsf{st}_{\mathrm{in}}, \mathsf{sym}_{\mathrm{in}})$ and $\mathsf{pos}_{\mathrm{out}} = \mathsf{pos}_{\mathrm{in}} + \beta$.
10. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{rej}}$ output $\perp$.
11. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ **and** $\alpha =$‘B’, output $\perp$.
    Else if $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ and $\alpha =$‘A’, output msg.
12. Compute $w_{\mathrm{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, \mathsf{sym}_{\mathrm{out}}, \mathsf{pos}_{\mathrm{in}}, aux)$.
13. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}))$.
14. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
15. Let $m_{\mathrm{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\mathrm{out}}, w_{\mathrm{out}}, \mathsf{pos}_{\mathrm{out}})$ and $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\mathrm{out}})$.
16. Output $\mathsf{pos}_{\mathrm{out}}, \mathsf{sym}_{\mathrm{out}}, \mathsf{st}_{\mathrm{out}}, w_{\mathrm{out}}, v_{\mathsf{out}}, \sigma_{\mathrm{out}}$.

Figure 28: Prog-0-i-a

Prog-0-i-b

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_A \in \mathcal{K}$, Punctured PRF key $K_B\{i-1\} \in \mathcal{K}_p$, verification key VK_C.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. If $t \neq i$, let $F(K_A, t-1) = r_A$, $F.\mathsf{eval}(K_B\{i-1\}, t-1) = r_B$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$, $\text{VK} = \text{VK}_{B,\text{rej}}$.
   Else $\text{VK} = \text{VK}_C$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\text{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $(t > t^*$ **or** $t \leq i-1)$ output $\perp$.
6. If $\alpha =$'-' **and** $t = i$ **and** $\mathsf{Verify\text{-}Spl}(\text{VK}, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
7. If $\alpha =$ '-' **and** $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
8. If $\alpha =$'-', output $\perp$.
9. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
11. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$'B', output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$'A', output msg.
12. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$.
13. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
14. Let $r'_A = F(K_A, t)$, $r'_B = F.\mathsf{eval}(K_B\{i-1\}, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
15. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
16. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 29: Prog-0-i-b

Hybrid $H_{i,c}$ This experiment is similar to $H_{i,b}$, except that r_C is chosen uniformly at random from $\{0,1\}^\lambda$. More formally, the challenger computes $K_B\{i-1\}$ 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 $H_{i,d}$ In this hybrid, the challenger chooses $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ as before. However, instead of hardwiring $\text{VK}_{C,\text{rej}}$, it hardwires VK_C.

Hybrid $H_{i,e}$ In this hybrid, the challenger uses a pseudorandom string to compute $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}})$. More formally, the challenger computes $r_C = F(K_B, i-1)$, $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$.

Hybrid $H_{i,f}$ This experiment corresponds to H_{i-1}.

Analysis Let $\mathsf{Adv}^{i,x}_{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in $H_{i,x}$, and let $\mathsf{Adv}^{i}_{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in H_i.

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

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

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

Proof. The only difference between $\mathsf{Prog}\text{-}0\text{-}i\text{-}a$ and $\mathsf{Prog}\text{-}0\text{-}i\text{-}b$ is that the latter uses a punctured PRF key $K_B\{i-1\}$ at steps 2 and 14. At step 2, functionality is preserved since the correct verification key is hardwired as VK_C in the hybrid. Next, note that step 14 functionality can possibly differ only if $t = i - 1$ and $\alpha =$'B'. However, by definition of the program, this case is not possible. ■

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

Proof. Note that both programs depend only on $K_B\{i-1\}$. As a result, we can replace $F(K_B, i-1)$ with a random value. From the security of puncturable PRFs, it follows that these two hybrids are computationally indistinguishable. ■

Claim A.5. Assuming $\mathcal{S}$ is a splittable signature scheme satisfying VK_{rej} indistinguishability (Definition 5.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{i,c}_{\mathcal{A}} - \mathsf{Adv}^{i,d}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

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 $\text{VK}_{C,\text{rej}}$, the experiments are indistinguishable. ■

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

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

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

Proof. The only difference between $H_{i,e}$ and $H_{i,f}$ is that $H_{i,e}$ uses a PRF key $K_B\{i-1\}$ punctured at $i-1$, while $H_{i,f}$ uses K_B itself. Using the correctness property of puncturable PRFs, we can argue that the programs output in $H_{i,e}$ and $H_{i,f}$ are functionally identical, and therefore $H_{i,e}$ and $H_{i,f}$ are computationally indistinguishable (implied by the security of $i\mathcal{O}$). ■

To conclude, for any PPT adversary $\mathcal{A}$, if $\mathcal{A}$ has advantage $\mathsf{Adv}^0_{\mathcal{A}}$ in Hyb_0 and $\mathsf{Adv}^1_{\mathcal{A}}$ in Hyb_1, then $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

■

A.2 Proof of Lemma 6.2

Proof Intuition Let us first note the differences between $\mathsf{Prog}\text{-}2\text{-}i$ and $\mathsf{Prog}'\text{-}2\text{-}i$.

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

In hybrid $\mathsf{Hyb}_{2,i}$ the challenger outputs a program $P_{2,i}$, while in hybrid $\mathsf{Hyb}'_{2,i}$, the challenger outputs a program $P'_{2,i}$. We need to show that $P_{2,i}$ and $P'_{2,i}$ are indistinguishable, even though their programs could differ at inputs corresponding to $i, i+1$. The first step in this direction is to transform $P_{2,i}$ into a program Q_1 that, at time $t = i+1$, ensures that if a signature passes the 'A' verification, then the message signed must be m_i - the correct input message. This is achieved using properties of splittable signatures.

Next, we enforce that if Q_1 verifies an 'A' type signature for m_i at time $i+1$, then the output state must not be q_{acc}. Note that during correct evaluation, we will not reach state q_{acc}, and therefore, this can be enforced using the accumulator security property. At this point, Q_1 accepts both 'A/B' type signatures at time $i+1$ and still outputs 'A/B' type signatures at time i. Let $\mathrm{VK}_A^{i+1}, \mathrm{VK}_B^{i+1}$ be the verification keys used at time $i+1$, and $\mathrm{SK}_A^i, \mathrm{SK}_B^i$ the secret keys used at time i. Using the properties of splittable signatures, we change it so that it uses only VK_A^{i+1} at step $i+1$, and only SK_A^i at step i. This completes our proof.

Formal Proof To prove Lemma 6.2, we will first define a sequence of hybrids $H_0, \ldots, H_{13}$, where H_0 corresponds to $\mathsf{Hyb}_{2,i}$ and H_{13} corresponds to $\mathsf{Hyb}'_{2,i}$.

Hybrid H_0 This experiment corresponds to $\mathsf{Hyb}_{2,i}$.

Hybrid H_1 In this experiment, the challenger punctures key K_A, K_B at input i, uses $F(K_A, i)$ and $F(K_B, i)$ to compute $(\mathrm{SK}_C, \mathrm{VK}_C)$ and $(\mathrm{SK}_D, \mathrm{VK}_D)$ respectively. More formally, it computes $K_A\{i\} \leftarrow F.\mathsf{puncture}(K_A, i)$, $r_C = F(K, i)$, $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$ and $K_B\{i\} \leftarrow F.\mathsf{puncture}(K_B, i)$, $r_D = F(K, i)$, $(\mathrm{SK}_D, \mathrm{VK}_D, \mathrm{VK}_{D,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_D)$.

It then hardwires $K_A\{i\}, K_B\{i\}, \mathrm{SK}_C, \mathrm{VK}_C, \mathrm{SK}_D, \mathrm{VK}_D$ in an altered program $P = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}1}\{K_A\{i\}$, $K_B\{i\}$, SK_C, VK_C, SK_D, VK_D, $m_i\}$(defined in Figure 30) and outputs its obfuscation. P is identical to $\mathsf{Prog}\text{-}2\text{-}i$, except that it uses a punctured PRF key $K_A\{i\}$ instead of K_A, and $K_B\{i\}$ instead of K_B. On input corresponding to i, P uses the hardwired keys.

Hybrid H_2 In this hybrid, the challenger chooses r_C, r_D uniformly at random instead of computing them using $F(K_A, i)$ and $F(K_B, i)$. In other words, the secret key/verification key pairs are sampled as $(\mathrm{SK}_C, \mathrm{VK}_C) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and $(\mathrm{SK}_D, \mathrm{VK}_D) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$.

Hybrid H_3 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 $(\sigma_{C,\text{one}}$, $\mathrm{VK}_{C,\text{one}}$, $\sigma_{C,\text{abo}}$, $\mathrm{VK}_{C,\text{abo}}) = \mathsf{Split}(\mathrm{SK}_C, m_i)$ and $(\sigma_{D,\text{one}}$, $\mathrm{VK}_{D,\text{one}}$, $\sigma_{D,\text{abo}}$, $\mathrm{VK}_{D,\text{abo}}) = \mathsf{Split}(\mathrm{SK}_D, m_i)$.

Prog-2-i-1

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, punctured PRF keys $K_A\{i\}, K_B\{i\} \in \mathcal{K}_p$, secret/verification keys $\mathrm{SK}_C, \mathrm{SK}_D, \mathrm{VK}_D, \mathrm{VK}_D$, message m_i.

Input: Time $t \in [T]$, position $\mathsf{pos}_{\text{in}} \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. If $t \neq i+1$, let $r_A = F.\mathsf{eval}(K_A\{i\}, t-1), r_B = F.\mathsf{eval}(K_B\{i\}, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
   Else set $\mathrm{VK}_A = \mathrm{VK}_C, \mathrm{VK}_B = \mathrm{VK}_D$.
3. Let $\alpha =$‘-’ and $m_{\text{in}} = (v_{\text{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$‘A’ .
5. If $\alpha =$‘-’ **and** $(t > t^*$ **or** $t \leq i)$ output $\perp$.
6. If $\alpha =$ ‘-’ **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$‘B’.
7. If $\alpha =$‘-’, output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$‘B’ output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$ ‘A’ **and** $t \leq i$ output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$‘A’ output msg.
11. Compute $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{out}}, aux)$.
12. Compute $v_{\text{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\text{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. If $t \neq i$, let $r'_A = F.\mathsf{eval}(K_A\{i\}, t), r'_B = F.\mathsf{eval}(K_B\{i\}, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
    Else set $\mathrm{SK}'_A = \mathrm{SK}_C, \mathrm{SK}'_B = \mathrm{SK}_D$.
14. Let $m_{\text{out}} = (v_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = i$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\text{out}})$.
    Else if $t = i$ **and** $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\text{out}}, \sigma_{\text{out}}$.

Figure 30: Prog-2-i-1

It then hardwires $\sigma_{C,\text{one}}$, $\mathrm{SK}_{D,\text{abo}}$ in $P = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}2}\{K_A\{i\}, K_B\{i\}, \sigma_{C,\text{one}}, \mathrm{VK}_C, \mathrm{SK}_{D,\text{abo}}, \mathrm{VK}_D, m_i\}$ (defined in Figure 31) and outputs an obfuscation of P. Note that the only difference between Prog-2-i-2 and Prog-2-i-1 is that Prog-2-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, Prog-2-i-2 outputs $\sigma_{C,\text{one}}$ if the outgoing message $m = m_i$, else it signs using $\mathrm{SK}_{C,\text{abo}}$.

Hybrid H_4 This hybrid is similar to the previous one, except that the challenger hardwires $\mathrm{VK}_{C,\text{one}}$ in Prog-2-i-2 instead of VK_C; that is, it computes $(\sigma_{C,\text{one}}, \mathrm{VK}_{C,\text{one}}, \sigma_{C,\text{abo}}, \mathrm{VK}_{C,\text{abo}}) = \mathsf{Split}(\mathrm{SK}_C, m_i)$ and $(\sigma_{D,\text{one}}, \mathrm{VK}_{D,\text{one}}, \sigma_{D,\text{abo}}, \mathrm{VK}_{D,\text{abo}}) = \mathsf{Split}(\mathrm{SK}_D, m_i)$ and outputs an obfuscation of $W_4 = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}2}\{K_A\{i\}, K_B\{i\}, \sigma_{C,\text{one}}, \mathrm{VK}_{C,\text{one}}, \mathrm{SK}_{D,\text{abo}}, \mathrm{VK}_D, m_i\})$.

Hybrid H_5 In this hybrid, the challenger hardwires $\mathrm{VK}_{D,\text{abo}}$ instead of VK_D. As in the previous hybrid, it uses Split to compute $(\sigma_{C,\text{one}}, \mathrm{VK}_{C,\text{one}}, \sigma_{C,\text{abo}}, \mathrm{VK}_{C,\text{abo}})$ and $(\sigma_{D,\text{one}}, \mathrm{VK}_{D,\text{one}}, \sigma_{D,\text{abo}}, \mathrm{VK}_{D,\text{abo}})$ from SK_C and SK_D respectively. However, it outputs an obfuscation of $W_5 = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}2}\{K_A\{i\}, K_B\{i\}, \sigma_{C,\text{one}}, \mathrm{VK}_{C,\text{one}}, \mathrm{SK}_{D,\text{abo}}, \mathrm{VK}_{D,\text{abo}}, m_i\}$.

Hybrid H_6 In this hybrid, the challenger outputs an obfuscation of $P = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}3}\{K_A\{i\}, K_B\{i\}, \sigma_{C,\text{one}}, \mathrm{VK}_{C,\text{one}}, \mathrm{SK}_{C,\text{abo}},\mathrm{VK}_{C,\text{abo}},m_i\}$ (described in Figure 32). This program performs extra checks before com-

Prog-2-i-2

Constants: Turing machine $M = \langle Q, \Sigma_{\rm tape}, \delta, q_0, q_{\rm acc}, q_{\rm rej}\rangle$, time bound T, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, punctured PRF keys $K_A\{i\}, K_B\{i\} \in \mathcal{K}_p$, constrained secret/verification keys $\sigma_{C,\rm one}, \mathrm{VK}_C, \mathrm{SK}_{{\rm abo},D}, \mathrm{VK}_D$, message m_i.

Input: Time $t \in [T]$, position $\mathsf{pos}_{\rm in} \in [T]$, symbol $\mathsf{sym}_{\rm in} \in \Sigma_{\rm tape}$, state $\mathsf{st}_{\rm in} \in Q$, accumulator value $w_{\rm in} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature $\sigma_{\rm in}$, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, \mathsf{sym}_{\rm in}, \mathsf{pos}_{\rm in}, \pi) = 0$ output $\perp$.
2. If $t \neq i+1$, let $r_A = F.\mathsf{eval}(K_A\{i\}, t-1), r_B = F.\mathsf{eval}(K_B\{i\}, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\rm rej})$ $= \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
   Else set $\mathrm{VK}_A = \mathrm{VK}_C, \mathrm{VK}_B = \mathrm{VK}_D$.
3. Let $\alpha =$'-' and $m_{\rm in} = (v_{\mathsf{in}}, \mathsf{st}_{\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in})$.
4. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** ($t > t^*$ **or** $t \leq i$) output $\perp$.
6. If $\alpha =$ '-' **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\rm in}, \sigma_{\rm in}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\rm out}, \mathsf{sym}_{\rm out}, \beta) = \delta(\mathsf{st}_{\rm in}, \mathsf{sym}_{\rm in})$ and $\mathsf{pos}_{\rm out} = \mathsf{pos}_{\rm in} + \beta$.
9. If $\mathsf{st}_{\rm out} = q_{\rm rej}$ output $\perp$.
10. If $\mathsf{st}_{\rm out} = q_{\rm acc}$ and $\alpha =$'B' output $\perp$.
    Else if $\mathsf{st}_{\rm out} = q_{\rm acc}$ **and** $\alpha =$ 'A' **and** $t \leq i$ output $\perp$.
    Else if $\mathsf{st}_{\rm out} = q_{\rm acc}$ **and** $\alpha =$'A' output msg.
11. Compute $w_{\rm out} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, \mathsf{sym}_{\rm out}, \mathsf{pos}_{\rm out}, aux)$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in}))$.
13. If $t \neq i$, let $r'_A = F.\mathsf{eval}(K_A\{i\}, t), r'_B = F.\mathsf{eval}(K_B\{i\}, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\rm rej}) =$ $\mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
    Else set $\mathrm{SK}'_A = \sigma_{C,\rm one}, \mathrm{SK}'_B = \mathrm{SK}_{{\rm abo},D}$.
14. Let $m_{\rm out} = (v_{\mathsf{out}}, \mathsf{st}_{\rm out}, w_{\rm out}, \mathsf{pos}_{\rm out})$.
    If $t = i$ **and** $m_{\rm out} = m_i$, $\sigma_{\rm out} = \sigma_{C,\rm abo}$.
    Else if $t = i$ **and** $m_{\rm out} \neq m_i$, $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl\text{-}abo}(\mathrm{SK}'_B, m_{\rm out})$.
    Else $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\rm out})$.
15. Output $\mathsf{pos}_{\rm out}, \mathsf{sym}_{\rm out}, \mathsf{st}_{\rm out}, w_{\rm out}, v_{\mathsf{out}}, \sigma_{\rm out}$.

Figure 31: Prog-2-i-2

puting the signature. In particular, the program additionally checks if the input corresponds to step $i+1$. If so, it checks whether $m_{\rm in} = m_i$ or not, and accordingly outputs either 'A' or 'B' type signature.

Hybrid H_7 In this hybrid, the challenger makes the accumulator 'read enforcing'. It computes the first i 'correct inputs' for the accumulator. Initially, the state is $\mathsf{st}_0 = q_0$. Let tape be a T dimensional vector, the first ℓ_{inp} entries of tape correspond to the input inp. The remaining are '␣'. Let $\mathsf{sym}_0 = \mathsf{tape}[0]$ and $\mathsf{pos}_0 = 0$. For $j = 1$ to i

1. Let $(\mathsf{st}_j, \mathsf{sym}_{w,j}, \beta) = \delta(\mathsf{st}_{j-1}, \mathsf{sym}_{j-1})$.
2. Set $\mathsf{tape}[\mathsf{pos}_{j-1}] = \mathsf{sym}_{w,j}$, $\mathsf{pos}_j = \mathsf{pos}_{j-1} + \beta$, $\mathsf{sym}_j = \mathsf{tape}[\mathsf{pos}_j]$.

Let $\mathsf{enf} = ((\mathsf{inp}_1, 0), \ldots, (\mathsf{inp}_{\ell_{\mathsf{inp}}}, \ell_{\mathsf{inp}} - 1), (\mathsf{sym}_{w,1}, \mathsf{pos}_0), \ldots, (\mathsf{sym}_{w,i}, \mathsf{pos}_{i-1}))$. The challenger computes $(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, \mathsf{enf}, \mathsf{pos}_i)$.

Hybrid H_8 In this hybrid, the challenger outputs an obfuscation of program $W_8 =$ Prog-2-i-4$\{K_A\{i\}$, $K_B\{i\}$, $\sigma_{C,\rm one}$, $\mathrm{VK}_{C,\rm one}$, $\mathrm{SK}_{D,\rm abo}$, $\mathrm{VK}_{D,\rm abo}$, $m_i\}$ (defined in Figure 33). This program outputs $\perp$ if on $(i+1)^{th}$ step, the input signature 'A' verifies, and the output state is $q_{\rm acc}$. Note that the accumulator is 'read enforced' in this hybrid.

Prog-2-i-3

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, punctured PRF keys $K_A\{i\}, K_B\{i\} \in \mathcal{K}_p$, constrained secret/verification keys $\sigma_{C,\text{one}}, \text{VK}_C, \text{SK}_{\text{abo},D}, \text{VK}_D$, message m_i.

Input: Time $t \in [T]$, position $\mathsf{pos}_{\text{in}} \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. If $t \neq i+1$, let $r_A = F.\mathsf{eval}(K_A\{i\}, t-1), r_B = F.\mathsf{eval}(K_B\{i\}, t-1)$. 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 $\text{VK}_A = \text{VK}_C, \text{VK}_B = \text{VK}_D$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $(t > t^*$ **or** $t \leq i)$ output $\perp$.
6. If $\alpha =$ '-' **and** $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$'B' output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$ 'A' **and** $t \leq i$ output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$'A' output msg.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{out}}, aux)$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. If $t \neq i$, let $r'_A = F.\mathsf{eval}(K_A\{i\}, t), r'_B = F.\mathsf{eval}(K_B\{i\}, t)$. 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 $\text{SK}'_A = \sigma_{C,\text{one}}, \text{SK}'_B = \text{SK}_{\text{abo},D}$.
14. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = i$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \sigma_{C,\text{abo}}$.
    Else if $t = i$ **and** $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl\text{-}abo}(\text{SK}'_B, m_{\text{out}})$.
    Else if $t = i+1$ **and** $m_{\text{in}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
    Else if $t = i+1$ **and** $m_{\text{in}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 32: Prog-2-i-3

Hybrid H_9 In this hybrid, the challenger uses normal setup for the accumulator related parameters; that is, it computes $(\text{PP}_{\mathsf{Acc}}, w_0, store_0) \leftarrow \mathsf{Setup\text{-}Acc}(1^\lambda, T)$. The remaining steps are exactly identical to the corresponding ones in the previous hybrid.

Hybrid H_{10} 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 $W_{10} = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}4}\{K_A\{i\}, K_B\{i\}$, $\sigma_{C,\text{one}}$, $\text{VK}_{C,\text{one}}$, $\text{SK}_{C,\text{abo}}$,$\text{VK}_{C,\text{abo}}$,$m_i\}$. Note that the hardwired keys for verification/signing (that is, $\sigma_{C,\text{one}}$, $\text{VK}_{C,\text{one}}$, $\text{SK}_{C,\text{abo}}$,$\text{VK}_{C,\text{abo}}$) 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_{11} In this hybrid, the challenger obfuscates a program Prog-2-i-5 (defined in Figure 34) which has a secret key, verification key pair hardwired, instead of the four components in Prog-2-i-4. More formally, the challenger chooses $(\text{SK}_C, \text{VK}_C) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and outputs an obfuscation of $W_{11} = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}5}\{K_A\{i\}$, $K_B\{i\}$, $\text{SK}_C, \text{VK}_C\}$.

Hybrid H_{12} In this hybrid, the challenger chooses the randomness r_C used to compute $(\text{SK}_C, \text{VK}_C)$ pseudorandomly; that is, it sets $r_C = F(K_A, i)$.

Prog-2-i-4

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, punctured PRF keys $K_A\{i\}, K_B\{i\} \in \mathcal{K}_p$, constrained secret/verification keys $\sigma_{C,\text{one}}, \mathrm{VK}_C, \mathrm{SK}_{\text{abo},D}, \mathrm{VK}_D$, message m_i.

Input: Time $t \in [T]$, position $\mathsf{pos}_{\text{in}} \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. If $t \neq i+1$, let $r_A = F.\mathsf{eval}(K_A\{i\}, t-1), r_B = F.\mathsf{eval}(K_B\{i\}, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
   Else set $\mathrm{VK}_A = \mathrm{VK}_C, \mathrm{VK}_B = \mathrm{VK}_D$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $(t > t^*$ **or** $t \leq i)$ output $\perp$.
6. If $\alpha =$ '-' **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$'B' output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$ 'A' **and** $\underline{t \leq i+1}$ output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$'A' output msg.
11. Compute $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{out}}, aux)$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. If $t \neq i$, let $r'_A = F.\mathsf{eval}(K_A\{i\}, t), r'_B = F.\mathsf{eval}(K_B\{i\}, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
    Else set $\mathrm{SK}'_A = \sigma_{C,\text{one}}, \mathrm{SK}'_B = \mathrm{SK}_{\text{abo},D}$.
14. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = i$ **and** $m_{\text{out}} = m_i$, $\sigma_{\text{out}} = \sigma_{C,\text{abo}}$.
    Else if $t = i$ **and** $m_{\text{out}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl\text{-}abo}(\mathrm{SK}'_B, m_{\text{out}})$.
    Else if $t = i+1$ **and** $m_{\text{in}} = m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\text{out}})$.
    Else if $t = i+1$ **and** $m_{\text{in}} \neq m_i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 33: Prog-2-i-4

Hybrid H_{13} This corresponds to the hybrid $\mathsf{Hyb}'_{2,i}$.

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

Claim A.8. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. The only difference between H_0 and H_1 is that H_0 uses program Prog-2-i, while H_1 uses Prog-2-i-1. From the correctness of puncturable PRFs, it follows that both programs have identical functionality for $t \neq i$. For $t = i$, the two programs have identical functionality because $(\mathrm{SK}_C, \mathrm{VK}_C)$ and $(\mathrm{SK}_D, \mathrm{VK}_D)$ are correctly computed using $F(K_A, i)$ and $F(K_B, i)$ respectively. Therefore, by the security of $i\mathcal{O}$, it follows that the obfuscations of the two programs are computationally indistinguishable. ∎

Claim A.9. Assuming F is a selectively secure puncturable PRF, for any adversary $\mathcal{A}$, $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. We will construct an intermediate experiment H, where r_C is chosen uniformly at random, while $r_D = F(K_B, i)$. Now, if an adversary can distinguish between H_1 and H, then we can construct a reduction

Prog-2-i-5

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, punctured PRF keys $K_A\{i\}, K_B\{i\} \in \mathcal{K}_p$, constrained secret/verification keys SK_C, VK_C, message m_i.

Input: Time $t \in [T]$, position $\mathsf{pos}_{\text{in}} \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. If $t \neq i+1$, let $r_A = F.\mathsf{eval}(K_A\{i\}, t-1), r_B = F.\mathsf{eval}(K_B\{i\}, t-1)$. 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 $\text{VK}_A = \text{VK}_C$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\text{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $(t > t^*$ **or** $t \leq i+1)$ output $\perp$.
6. If $\alpha =$ '-' **and** $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$'B' output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$ 'A' **and** $t \leq i+1$ output $\perp$.
    Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$'A' output msg.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{out}}, aux)$.
12. Compute $v_{\text{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\text{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. If $t \neq i$, let $r'_A = F.\mathsf{eval}(K_A\{i\}, t), r'_B = F.\mathsf{eval}(K_B\{i\}, t)$. 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 $\text{SK}'_A = \text{SK}_C$.
14. Let $m_{\text{out}} = (v_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = i$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
    Else if $t = i+1$ **and** $m_{\text{in}} = m_i$, $\mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
    Else if $t = i+1$ **and** $m_{\text{in}} \neq m_i$, $\mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\text{out}}, \sigma_{\text{out}}$.

Figure 34: Prog-2-i-5

algorithm that breaks the security of F. The reduction algorithm sends i as the challenge, and receives $K_A\{i\}, r$. It then uses r to compute $(\text{SK}_C, \text{VK}_C) = \mathsf{Setup\text{-}Spl}(1^\lambda; r)$. 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_1 and H with advantage ϵ, then $\mathcal{B}$ breaks the PRF security with advantage ϵ. ■

Claim A.10. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This follows from correctness property 2 of splittable signatures. This correctness property ensures that Prog-2-i-1 and Prog-2-i-2 have identical functionality. ■

Claim A.11. Assuming $\mathcal{S}$ satisfies VK_{one} indistinguishability (Definition 5.2), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}}| = \epsilon$. Then we can construct a reduction algorithm $\mathcal{B}$ that breaks the VK_{one} indistinguishability of $\mathcal{S}$. $\mathcal{B}$ sends m_i to the challenger. The challenger chooses $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$, $(\sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \text{SK}_{C,\text{abo}}, \text{VK}_{C,\text{abo}})$ and receives (σ, VK), where $\sigma = \sigma_{C,\text{one}}$ and $\text{VK} = \text{VK}_C$ or $\text{VK}_{C,\text{one}}$. It chooses the remaining components (including $\text{SK}_{D,\text{abo}}$ and

VK_D), and computes $\mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}2}\{K_A\{i\},\ K_B\{i\},\ \sigma,\ \mathrm{VK},\ \mathrm{SK}_{D,\mathrm{abo}},\ \mathrm{VK}_D,m_i\}$. Now, note that $\mathcal{B}$ perfectly simulates either H_4 or H_5, depending on whether the challenge message was a ■

Claim A.12. Assuming $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{abo}}$ indistinguishability (Definition 5.3), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^4_{\mathcal{A}} - \mathsf{Adv}^5_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This proof is similar to the previous one. If there exists an adversary $\mathcal{A}$ such that $\mathsf{Adv}^4_{\mathcal{A}} - \mathsf{Adv}^5_{\mathcal{A}} = \epsilon$, 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 A.13. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^5_{\mathcal{A}} - \mathsf{Adv}^6_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Let $P_0 = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}2}\{K_A\{i\}, K_B\{i\}, \sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{D,\mathrm{abo}}, \mathrm{VK}_{D,\mathrm{abo}}, m_i\}$ and $P_1 = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}3}\{K_A\{i\}, K_B\{i\}, \sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{C,\mathrm{abo}},\mathrm{VK}_{C,\mathrm{abo}},m_i\}$, 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 $i+1$. Fix any input $(i+1, m_{\mathrm{in}} = (v_{\mathrm{in}}, \mathsf{st}_{\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}), \mathsf{sym}_{\mathrm{in}}, \pi, aux$. 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 A.14. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^6_{\mathcal{A}} - \mathsf{Adv}^7_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This follows from Definition 4.1. Suppose, on the contrary, there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}^6_{\mathcal{A}} - \mathsf{Adv}^7_{\mathcal{A}}| = \epsilon$ 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 computes the first i tuples to be accumulated. It computes $(\mathsf{sym}_{w,j}, \mathsf{pos}_j)$ for $j \leq i$ as described in Hybrid H_7, and sends $(\mathsf{sym}_{w,j}, \mathsf{pos}_j)$ for $j < i$, and pos_i to the challenger, and receives $(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0)$. $\mathcal{B}$ 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 A.15. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^7_{\mathcal{A}} - \mathsf{Adv}^8_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Let $P_0 = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}3}\{K_A\{i\}, K_B\{i\}, \sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{C,\mathrm{abo}},\mathrm{VK}_{C,\mathrm{abo}},m_i\}$ and $P_1 = \mathsf{Prog\text{-}2\text{-}}i\mathsf{\text{-}4}\{K_A\{i\}, K_B\{i\}, \sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{C,\mathrm{abo}},\mathrm{VK}_{C,\mathrm{abo}},m_i\}$. We need to show that P_0 and P_1 have identical functionality. Note the only difference could be in the case where $t = i+1$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{C,\mathrm{one}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$ and the remaining inputs are such that $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$, 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, $w_{\mathrm{in}} = w_i$, $\mathsf{pos}_{\mathrm{in}} = \mathsf{pos}_i$, $\mathsf{st}_{\mathrm{in}} = \mathsf{st}_i$. Therefore, ($\mathsf{sym}_{\mathrm{in}} = \epsilon$ **or** $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, \mathsf{sym}_{\mathrm{in}}, w_i, \mathsf{pos}_i, \pi) = 1) \implies \mathsf{sym}_{\mathrm{in}} = \mathsf{sym}_i$, which implies $\mathsf{st}_{\mathrm{out}} = \mathsf{st}_{i+1}$. However, since M is not accepting, $\mathsf{st}_{i+1} \neq q_{\mathrm{acc}}$. Therefore, $t = i+1$ **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_{C,\mathrm{one}}, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$ **and** $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ cannot take place. ■

Claim A.16. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^8_{\mathcal{A}} - \mathsf{Adv}^9_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This step is a reversal of the step from H_6 to H_7, and therefore the proof of this claim is similar to that of Claim A.14. ∎

Claim A.17. Assuming $\mathcal{S}$ satisfies splitting indistinguishability (Definition 5.4), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^9_{\mathcal{A}} - \mathsf{Adv}^{10}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. We will use the splittable indistinguishability property (Definition 5.4) for this claim. Assume there is a PPT adversary $\mathcal{A}$ such that $|\mathsf{Adv}^9_{\mathcal{A}} - \mathsf{Adv}^{10}_{\mathcal{A}}| = \epsilon$. 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 $(\sigma_{\text{one}}, \text{VK}_{\text{one}}, \text{SK}_{\text{abo}}, \text{VK}_{\text{abo}})$, 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 Prog-2-i-4. It computes $K_A\{i\}, K_B\{i\}$, $\text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}$, m_i as described in hybrid H_9 and hardwires $\sigma_{\text{one}}, \text{VK}_{\text{one}}, \text{SK}_{\text{abo}}, \text{VK}_{\text{abo}}$ in the program. In this way, $\mathcal{B}$ can simulate either H_9 or H_{10}, and therefore, use $\mathcal{A}$'s advantage to break the splitting indistinguishability. ∎

Claim A.18. Assuming $\mathcal{S}$ satisfies splitting indistinguishability (Definition 5.4), for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{10}_{\mathcal{A}} - \mathsf{Adv}^{11}_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. This claim follows from correctness properties of $\mathcal{S}$. Note that the programs W_{10} and W_{11} can possibly differ only if $t = i + 1$. 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 $\mathsf{st}_{\text{in}} = q_{\text{acc}}$. Both programs output $\perp$.
2. $\mathsf{Verify\text{-}Spl}(\text{VK}_{C,\text{one}}, m_{\text{in}}, \sigma_{\text{in}}) = 1$ and $\mathsf{st}_{\text{out}} \neq q_{\text{acc}}$. In this case, W_{10} outputs $\mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$. Note that using correctness properties 3 and 5, we get that $m_{\text{in}} = m_i$, and therefore, W_{11} outputs $\mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
3. $\mathsf{Verify\text{-}Spl}(\text{VK}_{C,\text{one}}, m_{\text{in}}, \sigma_{\text{in}}) = 0$ but $\mathsf{Verify\text{-}Spl}(\text{VK}_{C,\text{abo}}, m_{\text{in}}, \sigma_{\text{in}}) = 1$. In this case, W_{10} sets $\alpha =$'B', and therefore the program outputs $\mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$. Using property 6, it follows that $m_{\text{in}} \neq m_i$, 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 A.19. Assuming F is a selectively secure puncturable PRF scheme, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{11}_{\mathcal{A}} - \mathsf{Adv}^{12}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. The proof of this claim is identical to the proof of Claim A.9.

∎

Claim A.20. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{12}_{\mathcal{A}} - \mathsf{Adv}^{13}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This proof is identical to the proof of Claim A.8; it follows directly from the correctness of puncturable PRFs. ∎

A.3 Proof of Lemma 6.3

Proof Intuition The only difference between Prog'-2-i and Prog-2-$i+1$ is for inputs corresponding to $t = i + 1$. Both programs accept only 'A' type signatures at $t = i + 1$, and both output either 'A' or 'B' type signatures. However, Prog'-2-i outputs an 'A' type signature at $t = i + 1$ iff the input message m_{in} is the correct one (that is, m_i). On the other hand, Prog-2-$i+1$ checks if the outgoing message m_{out} is the correct one (that is, m_{i+1}). Therefore, we need to argue that the adversary cannot find an input such that

$m_{\text{in}} \neq m_i$ but $m_{\text{out}} = m_{i+1}$, or $m_{\text{in}} = m_i$ but $m_{\text{out}} \neq m_{i+1}$. To show this, we will use the enforcement properties of the accumulator and the iterator.

Let Q_1 be the program which outputs an 'A' signature at $t = i+1$ iff $m_{\text{in}} = m_{i+1}$ and the state, position, iterator values output are also the correct ones. By read enforcing the accumulator, we can ensure that if the accumulated value verifies, then the input symbol sym_{in} is the correct one. If $m_{\text{in}} = m_i$, then the input state, symbol and position are all correct, implying that the output state, symbol and position are also correct. So, no adversary can distinguish between Prog'-2-i and Q_1.

But it is possible that the adversary could send as input a 'fake' auxiliary input for the accumulator that results in the final accumulated value being wrong, even though $m_{\text{in}} = m_i$ and the output state, symbol and position are correct. To avoid this, we use write enforcing of the accumulator. Let Q_2 be a program which checks whether $m_{\text{in}} = m_i$ and $m_{\text{out}} = m_{i+1}$. From the above discussion, it follows that Q_1 and Q_2 are computationally indistinguishable.

Finally, we need to remove the $m_{\text{in}} = m_i$ condition from Q_2. For this, we use the enforcing property of the iterator. This ensures that $m_{\text{in}} = m_i$ and $m_{\text{out}} = m_{i+1}$ iff $m_{out} = m_{i+1}$. This completes our proof.

Formal Proof We will first define a sequence of hybrid experiments $H_0, \ldots, H_8$, where H_0 corresponds to $\mathsf{Hyb}'_{2,i}$ and H_9 corresponds to $\mathsf{Hyb}_{2,i+1}$.

Hybrid H_0 This corresponds to $\mathsf{Hyb}'_{2,i}$.

Hybrid H_1 In this hybrid, the challenger uses 'read enforced' setup for the accumulator. The challenger computes the first $\ell_{\mathsf{inp}} + i$ 'correct tuples' for the accumulator. Initially, the state is $\mathsf{st}_0 = q_0$. Let tape be a T dimensional vector, the first ℓ_{inp} entries of tape correspond to the input inp. The remaining are '␣'. Let $\mathsf{sym}_0 = \mathsf{tape}[0]$ and $\mathsf{pos}_0 = 0$. For $j = 1$ to i

1. Let $(\mathsf{st}_j, \mathsf{sym}_{w,j}, \beta) = \delta(\mathsf{st}_{j-1}, \mathsf{sym}_{j-1})$.
2. Set $\mathsf{tape}[\mathsf{pos}_{j-1}] = \mathsf{sym}_{w,j}$, $\mathsf{pos}_j = \mathsf{pos}_{j-1} + \beta$, $\mathsf{sym}_j = \mathsf{tape}[\mathsf{pos}_j]$.

Let $\mathsf{enf} = ((\mathsf{inp}[0], 0), \ldots, (\mathsf{inp}[\ell_{\mathsf{inp}} - 1], \ell_{\mathsf{inp}} - 1), (\mathsf{sym}_{w,1}, \mathsf{pos}_0), \ldots, (\mathsf{sym}_{w,i}, \mathsf{pos}_{i-1}))$. The challenger computes $(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, \mathsf{enf}, \mathsf{pos}_i)$. The remaining steps are same as in the previous hybrid.

Hybrid H_2 In this hybrid, the challenger uses program P_2 (defined in Figure 35), which is similar to Prog'-2-i. However, in addition to checking if $m_{\text{in}} = m_i$, it also checks if $(v_{\mathsf{out}}, \mathsf{pos}_{\text{out}}, \mathsf{st}_{\text{out}}) = (v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1})$.

Hybrid H_3 In this experiment, the challenger uses normal setup instead of 'read enforced' setup for the accumulator.

Hybrid H_4 In this hybrid, the challenger 'write enforces' the accumulator. As in hybrid H_1, the challenger computes the first $\ell_{\mathsf{inp}}+i+1$ 'correct tuples' to be accumulated. Let $\mathsf{sym}_{w,j}, \mathsf{pos}_j$ be the symbol output and the position after the j^{th} step. The challenger computes $(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Write}(1^\lambda, T, \mathsf{enf})$, where $\mathsf{enf} = ((\mathsf{inp}[0], 0), \ldots, (\mathsf{inp}[\ell_{\mathsf{inp}} - 1], \ell_{\mathsf{inp}} - 1), (\mathsf{sym}_{w,1}, \mathsf{pos}_0), \ldots, (\mathsf{sym}_{w,i}, \mathsf{pos}_{i-1}), (\mathsf{sym}_{w,i+1}, \mathsf{pos}_i))$. The remaining computation is same as in previous step.

Hybrid H_5 In this experiment, the challenger outputs an obfuscation of $P_5 = P_5\{i, K_A, K_B, m_i, m_{i+1}\}$, which is similar to P_2. However, on input where $t = i + 1$, before computing signature, it also checks if $w_{\text{out}} = w_{i+1}$. Therefore, it checks whether $m_{\text{in}} = m_i$ and $m_{\text{out}} = m_{i+1}$.

Hybrid H_6 This experiment is similar to the previous one, except that the challenger uses normal setup for accumulator instead of 'enforcing write'.

P_2

Constants: Turing machine $M = \langle Q, \Sigma_{\mathrm{tape}}, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$, message m_i, iterator value v_{i+1}, position $\mathsf{pos}_{i+1} \in [T]$, state $\mathsf{st}_{i+1} \in Q$.

Input: Time $t \in [T]$, position $\mathsf{pos}_{\mathrm{in}} \in [T]$, symbol $\mathsf{sym}_{\mathrm{in}} \in \Sigma'_{\mathrm{tape}}$, state $\mathsf{st}_{\mathrm{in}} \in Q$, accumulator value $w_{\mathrm{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, \mathsf{sym}_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $\alpha =$'-' and $m_{\mathrm{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $(t > t^*$ **or** $t \leq i+1)$ output $\perp$.
6. If $\alpha \neq$ 'A' **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\mathrm{out}}, \mathsf{sym}_{\mathrm{out}}, \beta) = \delta(\mathsf{st}_{\mathrm{in}}, \mathsf{sym}_{\mathrm{in}})$ and $\mathsf{pos}_{\mathrm{out}} = \mathsf{pos}_{\mathrm{in}} + \beta$.
9. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ and $\alpha =$'B' output $\perp$.
11. Else if $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ **and** $\alpha =$ 'A' **and** $t \leq i+1$ output $\perp$.
12. Else if $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ **and** $\alpha =$'A' output msg.
13. Compute $w_{\mathrm{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, \mathsf{sym}_{\mathrm{out}}, \mathsf{pos}_{\mathrm{in}}, aux)$.
14. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}))$.
15. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
16. Let $m_{\mathrm{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\mathrm{out}}, w_{\mathrm{out}}, \mathsf{pos}_{\mathrm{out}})$.
    If $t = i+1$ **and** $m_{\mathrm{in}} = m_i$ <u>**and** $(v_{\mathsf{out}}, \mathsf{pos}_{\mathrm{out}}, \mathsf{st}_{\mathrm{out}}) = (v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1})$</u>, $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\mathrm{out}})$.
    Else if $t = i+1$ **and** <u>$(m_{\mathrm{in}} \neq m_i$ **or** $(v_{\mathsf{out}}, \mathsf{pos}_{\mathrm{out}}, \mathsf{st}_{\mathrm{out}}) \neq (v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1}))$</u>, $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\mathrm{out}})$.
    Else $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\mathrm{out}})$.
17. Output $\mathsf{pos}_{\mathrm{out}}, \mathsf{sym}_{\mathrm{out}}, \mathsf{st}_{\mathrm{out}}, w_{\mathrm{out}}, v_{\mathsf{out}}, \sigma_{\mathrm{out}}$.

Figure 35: P_2

Hybrid H_7 This experiment is similar to the previous one, except that the challenger uses enforced setup for iterator instead of normal setup. It first computes $\mathrm{PP}_{\mathsf{Acc}}, w_0, store_0$ 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. $(\mathsf{sym}_j, \pi_j) = \mathsf{Prep\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, store_{j-1}, \mathsf{pos}_{j-1})$.
2. Let $(\mathsf{st}_j, \mathsf{sym}_{w,j}, \beta) = \delta(\mathsf{st}_{j-1}, \mathsf{sym}_{j-1})$.
3. $aux_j = \mathsf{Prep\text{-}Write}(\mathrm{PP}_{\mathsf{Acc}}, store_{j-1}, \mathsf{pos}_{j-1})$.
4. $w_j = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{j-1}, \mathsf{sym}_{w,j}, \mathsf{pos}_{j-1}, aux_j)$.
5. $store_j = \mathsf{Write\text{-}Store}(\mathrm{PP}_{\mathsf{Acc}}, store_{j-1}, \mathsf{pos}_{j-1}, \mathsf{sym}_{w,j})$.
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 $(\mathrm{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{Setup\text{-}Itr\text{-}Enforce}(1^\lambda, T, \mathsf{enf})$. The remaining hybrid proceeds as the previous one.

Hybrid H_8 In this experiment, the challenger outputs an obfuscation of $P_8 = P_8\{i, K_A, K_B, m_{i+1}\}$ (defined in Figure 37), which is similar to P_5, except that it only checks if $m_{\mathrm{out}} = m_{i+1}$.

Hybrid H_9 This corresponds to $\mathsf{Hyb}_{2,i+1}$. The only difference between this experiment and the previous one is that this uses normal Setup for iterator.

P_5

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$, message m_i, m_{i+1}.

Input: Time $t \in [T]$, position $\mathsf{pos}_{\text{in}} \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma'_{\text{tape}}$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $(t > t^*$ **or** $t \leq i+1)$ output $\perp$.
6. If $\alpha \neq$ 'A' **and** $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$'B' output $\perp$.
11. Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$ 'A' **and** $t \leq i+1$ output $\perp$.
12. Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$'A' output msg.
13. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$.
14. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
15. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
16. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = i+1$ **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 $t = i+1$ **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 $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
17. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 36: P_5

A.3.1 Analysis

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

Claim A.21. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This proof is identical to the proof of Claim A.14; it follows from Read Setup indistinguishability (Definition 4.1) of Acc. ∎

Claim A.22. Assuming Acc is Read enforcing (Definition 4.3) and $i\mathcal{O}$ 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. In order to prove this claim, it suffices to show that $P_0 = \mathsf{Prog}'\text{-}2\text{-}i\{i, M, T, \mathsf{msg}_b, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, K_A, K_B, m_i\}$ and $P_1 = \mathsf{Prog}\text{-}2\text{-}i\text{-}b\{i, M, T, \mathsf{msg}_b, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, K_A, K_B, m_i, v_{i+1}, \mathsf{pos}_{i+1}, \mathsf{st}_{i+1}\}$ 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 4.3 and the definition of H_1/H_2, it follows that if $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_i, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 1$, 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 A.23. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

P_8

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, message msg, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$, message m_{i+1}.

Input: Time $t \in [T]$, position $\mathsf{pos}_{\text{in}} \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma'_{\text{tape}}$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $\alpha =$'-' and $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A' .
5. If $\alpha =$'-' **and** $(t > t^*$ **or** $t \leq i+1)$ output $\perp$.
6. If $\alpha \neq$ 'A' **and** $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$, set $\alpha =$'B'.
7. If $\alpha =$'-', output $\perp$.
8. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ and $\alpha =$'B' output $\perp$.
11. Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$ 'A' **and** $t \leq i+1$ output $\perp$.
12. Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $\alpha =$'A' output msg.
13. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$.
14. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
15. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
16. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = i+1$ **and** $\underline{m_{\text{out}} = m_{i+1}}$ set $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
    Else if $t = i+1$ **and** $\underline{(m_{\text{out}} \neq m_{i+1})}$, set $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
17. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 37: P_8

Proof. This step is a reversal of the step from H_0 to H_1; its proof is identical to that of Claim A.8. ∎

Claim A.24. Assuming Acc satisfies indistinguishability of Write Setup (Definition 4.2), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This proof follows from the Write Setup indistinguishability (Definition 4.2) of Acc. Suppose there exists an adversary $\mathcal{A}$ such that $\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}} = \epsilon$. We will construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the Write Setup indistinguishability of Acc. $\mathcal{B}$ first computes the first $\ell_{\mathsf{inp}} + i + 1$ tuples to be accumulated - $\mathsf{enf} = ((\mathsf{inp}[0], 0), \ldots, (\mathsf{inp}[\ell_{\mathsf{inp}} - 1], \ell_{\mathsf{inp}} - 1), (\mathsf{sym}_{w,1}, \mathsf{pos}_0), \ldots, (\mathsf{sym}_{w,i+1}, \mathsf{pos}_i))$. Next, it sends enf to the challenger, and receives $\text{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0$. The remaining encoding computation is identical in both hybrids, and therefore, $\mathcal{B}$ can simulate it perfectly using $\text{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0$. In this manner, $\mathcal{B}$ can perfectly simulate either H_3 or H_4, depending on the challenger's input, and then use $\mathcal{A}$'s response to win the security game with non-negligible advantage. ∎

Claim A.25. Assuming Acc is Write enforcing (Definition 4.4) and $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^4_{\mathcal{A}} - \mathsf{Adv}^5_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

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 4.4). Using the Write enforcing property, we can conclude that $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_i, \mathsf{sym}_{w,i+1}, \mathsf{pos}_i, aux) \implies w_{\text{out}} =$

w_{i+1} or $w_{\text{out}} = Reject$. In either case, we get that the functionality of P_0 and P_1 is identical, therefore implying that their obfuscations are indistinguishable. ∎

Claim A.26. Assuming Acc satisfies indistinguishability of Write Setup (Definition 4.2), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^5_{\mathcal{A}} - \mathsf{Adv}^6_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This step is a reversal of the step from H_3 to H_4; its proof is identical to that of Claim A.24. ∎

Claim A.27. Assuming Itr satisfies indistinguishability of Setup (Definition 3.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^6_{\mathcal{A}} - \mathsf{Adv}^7_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}^6_{\mathcal{A}} - \mathsf{Adv}^7_{\mathcal{A}}|$ is non-negligible. We will construct an algorithm $\mathcal{B}$ that breaks the Setup indistinguishability of Itr (Definition 3.1). $\mathcal{B}$ 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 $\text{PP}_{\mathsf{Itr}}, v_0$ 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 $|\mathsf{Adv}^6_{\mathcal{A}} - \mathsf{Adv}^7_{\mathcal{A}}|$ is non-negligible, $\mathcal{B}$'s advantage is also non-negligible. ∎

Claim A.28. Assuming Itr is enforcing (Definition 3.2) and $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^7_{\mathcal{A}} - \mathsf{Adv}^8_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. In order to prove this claim, we need to argue that $P_5 = P_5\{i, K_A, K_B, m_i, m_{i+1}\}$ 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 16: 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, $\text{PP}_{\mathsf{Itr}}, v_0$ 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 A.29. Assuming Itr satisfies indistinguishability of Setup (Definition 3.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^8_{\mathcal{A}} - \mathsf{Adv}^9_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This is a reversal of the step from H_7 to H_8, and its proof is similar to that of Claim A.27. ∎

A.4 Proof of Lemma 6.4

Proof Intuition The only differences between $\mathsf{Prog}'\text{-}2\text{-}t^*\!-\!1\{M, T, t^*, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, \mathsf{msg}_b, K_A, K_B, m_{t^*-1}\}$ and $\mathsf{Prog}\text{-}3\{M, T, t^*, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, \mathsf{msg}_b, K_A, K_B\}$ are:

1. $\mathsf{Prog}\text{-}3$ does not have the condition '$(t > t^*$ **or** $t \leq t^*)$' in the verification step, since this always holds true.
2. $\mathsf{Prog}\text{-}3$ never outputs an 'A' type signature on inputs corresponding to $t = t^*$. $\mathsf{Prog}'\text{-}2\text{-}t^* - 1$, by definition, can output an 'A' type signature if $m_{\text{in}} = m_{t^*-1}$. Intuitively, $\mathsf{Prog}'\text{-}2\text{-}t^* - 1$ never reaches to that point, since if $m_{\text{in}} = m_{t^*} - 1$, then the next state should be q_{rej}. To enforce this, we use Read-Enforce Setup for the accumulator.

Formal Proof We will now describe a sequence of hybrids, where H_0 corresponds to $\mathsf{Prog}'\text{-}2\text{-}t^* - 1\{M, T, t^*, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, \mathsf{msg}_b, K_A, K_B, m_{t^*-1}\}$ and H corresponds to $\mathsf{Prog}\text{-}3\{M, T, t^*, \text{PP}_{\mathsf{Acc}}, \text{PP}_{\mathsf{Itr}}, \mathsf{msg}_b, K_A, K_B\}$.

Hybrid H_1 In this hybrid, the challenger computes the parameters for the accumulator using read-enforced setup at position pos_{t^*-1}. It first computes the first t^*-1 'correct inputs' to be accumulated. Initially, the state is $\mathsf{st}_0 = q_0$. Let tape be a T dimensional vector, the first ℓ_{inp} entries of tape correspond to the input inp. The remaining are '␣'. Let $\mathsf{sym}_0 = \mathsf{tape}[0]$ and $\mathsf{pos}_0 = 0$. For $j = 1$ to t^*-1

1. Let $(\mathsf{st}_j, \mathsf{sym}_{w,j}, \beta) = \delta(\mathsf{st}_{j-1}, \mathsf{sym}_{j-1})$.
2. Set $\mathsf{tape}[\mathsf{pos}_{j-1}] = \mathsf{sym}_{w,j}$, $\mathsf{pos}_j = \mathsf{pos}_{j-1} + \beta$, $\mathsf{sym}_j = \mathsf{tape}[\mathsf{pos}_j]$.

Let $\mathsf{enf} = ((\mathsf{inp}[0], 0), \dots, (\mathsf{inp}[\ell_{\mathsf{inp}}-1], \ell_{\mathsf{inp}}-1), (\mathsf{sym}_{w,1}, \mathsf{pos}_0), \dots, (\mathsf{sym}_{w,t^*-1}, \mathsf{pos}_{t^*-2}))$ and let $(\mathrm{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}(1^\lambda, T, \mathsf{enf}, \mathsf{pos}_{t^*-1})$. The remaining hybrid proceeds as Hyb'_{2,t^*-1}.

Hybrid H_2 In this hybrid, the challenger outputs an obfuscation of $W_2 = \mathsf{Prog\text{-}3}\{M, T,\ t^*,\ \mathrm{PP}_{\mathsf{Acc}},\ \mathrm{PP}_{\mathsf{Itr}},\ \mathsf{msg}_b,\ K_A,\ K_B\}$.

Hybrid H_3 In this program, the challenger uses $\mathsf{Setup\text{-}Acc}$ for Acc instead of using $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}$. Note that this corresponds to Hyb_3.

A.4.1 Analysis

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

Claim A.30. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. This proof is identical to the proof of Claim A.14; it follows from Read Setup indistinguishability (Definition 4.1) of Acc. ∎

Claim A.31. 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. To prove this claim, we need to argue that $W_1 = \mathsf{Prog}'\text{-}2\text{-}t^*-1\{M, T,\ t^*,\ \mathrm{PP}_{\mathsf{Acc}},\ \mathrm{PP}_{\mathsf{Itr}},\ \mathsf{msg}_b,\ K_A,\ K_B,\ m_{t^*-1}\}$ and $W_2 = \mathsf{Prog\text{-}3}\{M, T,\ t^*,\ \mathrm{PP}_{\mathsf{Acc}},\ \mathrm{PP}_{\mathsf{Itr}},\ \mathsf{msg}_b,\ K_A,\ K_B\}$ 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 A.32. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. This step is a reversal of the step from H_0 to H_1; the proof is identical to the proof of Claim A.14. ∎

A.5 Proof of Lemma 6.5

Proof Intuition Note that Prog-3 does not output 'A' type signatures at time $t = t^*$. As a result, it is possible to modify the program so that the adversary has no information about the 'A' type secret key used at step t^*. Having done so, we can then replace the 'A' verification key used at step $t^* + 1$ with a reject-verification key that always outputs $\perp$. This ensures the program outputs $\perp$ at $t = t^* + 1$.

Continuing in this manner, suppose we have a program that outputs $\perp$ for all $t^* < t \leq i$. Then, we can modify it to remove the 'A' type secret key used at time $t = i$, and therefore replacing a normal 'A' type verification key at step $i + 1$ with a reject verification key. In this manner, we can get to Prog-4, which outputs $\perp$ for all $t > t^*$.

Formal Proof We will define $\tilde{t} = T - t^* + 1$ hybrids $H_0, \ldots, H_{\tilde{t}}$, and show that they are computationally indistinguishable.

H_i In this hybrid, the challenger outputs an obfuscation of Prog-3-$i\{i, \mathsf{msg}_b, K_A, K_B, m_{t^*-1}\}$ (defined in Figure 38).

Prog-3-i

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time t^*, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_A, K_B \in \mathcal{K}$, m_{t^*-1}.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. Let $F(K_A, t-1) = r_A, F(K_B, t-1) = r_B$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$, $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_B)$.
3. Let $\alpha =$'A' and $m_{\text{in}} = (v_{\text{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $t^* < t \leq i$ output $\perp$.
5. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
6. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
7. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
8. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output msg.
9. Compute $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
10. Compute $v_{\text{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\text{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
11. Let $r'_A = F(K_A, t), r'_B = F(K_B, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
12. Let $m_{\text{out}} = (v_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = t^*$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\text{out}})$.
13. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\text{out}}, \sigma_{\text{out}}$.

Figure 38: Prog-3-i

Clearly, programs Prog-3 and Prog-3-t^* are functionally identical, and therefore Hyb_3 and H_{t^*} are computationally indistinguishable. In order to show that H_i and H_{i+1} are computationally indistinguishable, we will define intermediate hybrid experiments $H_{i,a}, \ldots, H_i$, such that $H_{i,a}$ corresponds to H_i and $H_{i,f}$ corresponds to H_{i+1}.

$H_{i,a}$ This corresponds to H_i.

$H_{i,b}$ In this hybrid, the challenger first punctures the PRF key K_A at input i; that is, $K_A\{i\} \leftarrow F.\mathsf{puncture}(K_A, i)$. Next, it computes $r_C = F(K_A, i)$, $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$ and finally, outputs an obfuscation of $P_{i,b} = \mathsf{Prog\text{-}3\text{-}}i\text{-}b\{i, \mathsf{msg}_b,\ K_A,\ K_B\{i\},\ m_{t^*-1},\ \mathrm{VK}_C\}$ (defined in Figure 39). It has verification key VK_C hardwired.

Prog-3-i-b

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}\rangle$, time bound T, halt-time t^*, message msg, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Punctured PRF keys $K_A\{i\} \in \mathcal{K}_p$, Puncturable PRF key $K_B \in \mathcal{K}$, VK.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\text{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\text{in}} \in [T]$, state $\mathsf{st}_{\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{in}}, \mathsf{pos}_{\text{in}}, \pi) = 0$ output $\perp$.
2. If $t \neq i+1$, let $r_A = F.\mathsf{eval}(K_A\{i\}, t-1), r_B = F(K_B, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_A)$.
   Else let $\mathrm{VK}_A = \mathrm{VK}$.
3. Let $\alpha =$‘A’ and $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
4. If $t^* < t \leq i$ output $\perp$.
5. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
6. Let $(\mathsf{st}_{\text{out}}, \mathsf{sym}_{\text{out}}, \beta) = \delta(\mathsf{st}_{\text{in}}, \mathsf{sym}_{\text{in}})$ and $\mathsf{pos}_{\text{out}} = \mathsf{pos}_{\text{in}} + \beta$.
7. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output $\perp$.
8. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ **and** $t \leq t^*$ output $\perp$.
   Else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output msg.
9. Compute $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, \mathsf{sym}_{\text{out}}, \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
10. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
11. Let $r'_A = F.\mathsf{eval}(K_A\{i\}, t), r'_B = F(K_B, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_A)$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r'_B)$.
12. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = t^*$, $\sigma_{\text{out}} = \mathsf{Setup\text{-}Spl}(\mathrm{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\text{out}})$.
13. Output $\mathsf{pos}_{\text{out}}, \mathsf{sym}_{\text{out}}, \mathsf{st}_{\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 39: Prog-3-i-b

$H_{i,c}$ 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 Prog-3-i-b.

$H_{i,d}$ 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 Prog-3-i-b; that is, it outputs an obfuscation of $\mathsf{Prog\text{-}3\text{-}}i\text{-}b\{i, \mathsf{msg}_b,$ $K_A,\ K_B\{i\},\ m_{t^*-1},\ \mathrm{VK}_{C,\mathrm{rej}}\}$.

$H_{i,e}$ In this hybrid, the challenger chooses $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}})$ using $F(K, i)$. It computes $r_C = F(K, i)$, $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$, $i\mathcal{O}(\mathsf{Prog\text{-}3\text{-}}i\text{-}b\{i, \mathsf{msg}_b,\ K_A,\ K_B\{i\},\ m_{t^*-1},\ \mathrm{VK}_{C,\mathrm{rej}}\})$.

$H_{i,f}$ This hybrid corresponds to H_{i+1}.

A.5.1 Analysis

We will first show that H_i and H_{i+1} are computationally indistinguishable. Let $\mathsf{Adv}^i_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in H_0.

Claim A.33. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{i,a}_{\mathcal{A}} - \mathsf{Adv}^{i,b}_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. The only difference between $H_{i,a}$ and $H_{i,b}$ is that $H_{i,a}$ outputs an obfuscation of $\mathsf{Prog\text{-}3\text{-}}i$, while $H_{i,b}$ outputs an obfuscation of $\mathsf{Prog\text{-}3\text{-}}i\text{-}b$. $\mathsf{Prog\text{-}3\text{-}}i$ uses puncturable PRF key K_A, while $\mathsf{Prog\text{-}3\text{-}}i\text{-}b$ uses punctured key $K_A\{i\}$ punctured at i. $\mathsf{Prog\text{-}3\text{-}}i\text{-}b$ also has verification key VK_C hardwired, which is computed using $F(K_A, i)$. For $t \neq i+1$, both programs have identical functionality (this follows from the correctness of puncturable PRFs). For $t = i+1$, the verification part is identical, since VK_C hardwired is computed correctly. Also, note that the corresponding secret key is not required at $t = i$ (for $t = i$, 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 A.34. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{i,b}_{\mathcal{A}} - \mathsf{Adv}^{i,c}_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

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

Claim A.35. Assuming $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{rej}}$ indistinguishability, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{i,c}_{\mathcal{A}} - \mathsf{Adv}^{i,d}_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

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 $\mathsf{Adv}^{i,c}_{\mathcal{A}} - \mathsf{Adv}^{i,d}_{\mathcal{A}}$ 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 $\mathsf{Prog\text{-}3\text{-}}i\text{-}b$. 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 $\mathsf{Adv}^{i,c}_{\mathcal{A}} - \mathsf{Adv}^{i,d}_{\mathcal{A}}$ is non-negligible, $\mathcal{B}$'s advantage is also non-negligible. ∎

Claim A.36. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{i,d}_{\mathcal{A}} - \mathsf{Adv}^{i,e}_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. This step is the reverse of the step from $H_{i,b}$ to $H_{i,c}$; the proof follows from the security of puncturable PRFs. ∎

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

Proof. The only differences between the programs $\mathsf{Prog\text{-}3\text{-}}i\text{-}b$ and $\mathsf{Prog\text{-}3\text{-}}i+1$ are:

1. $\mathsf{Prog\text{-}3\text{-}}i\text{-}b$ uses a punctured PRF key $K_A\{i\}$, and has the reject-verification key computed using $F(K_A, i)$. As a result, it outputs $\perp$ for all inputs corresponding to $t = i+1$. $\mathsf{Prog\text{-}3\text{-}}i+1$, on the other hand, uses a puncturable PRF key K_A, and for inputs corresponding to $t = i+1$, 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, $H_{i,e}$ and $H_{i,f}$ are computationally indistinguishable. ∎

B Proofs for Section 7

B.1 Proof Outline for Lemma 7.1

Proof Intuition The main differences between Prog and $\mathsf{Prog\text{-}1}$ is that $\mathsf{Prog\text{-}1}$ has the halt-time t^* and the correct output b^* also hardwired, along with other constants. It outputs $\perp$ for all inputs corresponding to $t > t^*$. At t^*, it checks if the input passes the verification of accumulator and the verification of signature. If so, it outputs b^*, without decrypting the ciphertext. In order to show that Prog and $\mathsf{Prog\text{-}1}$ are computationally indistinguishable, we will break this into two big steps. First, we will modify Prog into a program P_{abort}.

P_{abort} is identical to Prog, except that it outputs b^* if the signature and accumulator verification passes. The next step is to transform P_{abort} so that it aborts for all $t > t^*$.

For the first step, our strategy is very similar to the proof of Theorem 6.1[5]. For the second step, our approach is very similar to the approach in the proof of Lemma 6.5. Note that at step t^*, P_{abort} does not output an 'A' signature. As a result, we can replace the verification key at step t^*+1 with a 'reject' verification key. Continuing this way, we can ensure that it is fine to output $\perp$ for all $t > t^*$.

Proof Outline We will first define hybrid experiments H_{int}, H'_{int} and H_{abort}.

Hybrid H_{int} In this hybrid, the challenger first computes the correct message m_{t^*-1} output at time t^*-1. Next, it outputs an obfuscation of $P_{int} = P_{int}\{t^*, K_E, K_A, K_B, m_{t^*-1}\}$ (defined in Figure 40) which has m_{t^*-1} hardwired. It accepts only 'A' type signatures. However, at $t = t^*-1$, it checks if the outgoing message is m_{t^*-1}. If so, it outputs an 'A' type signature, else it outputs a 'B' type signature.

Program P_{int}

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \in [T]$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, message m_{t^*-1}.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
3. Let $r_{S,A} = F(K_A, t-1)$, $r_{S,B} = F(K_B, t-1)$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$ and $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,B})$.
4. Let $m_{\text{in}} = (v_{\text{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
5. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
6. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
7. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
8. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
9. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0, else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
10. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\text{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. Let $r'_{S,A} = F(K_A, t)$, $r'_{S,B} = F(K_B, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
14. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = t^*-1$ **and** $m_{\text{out}} = m_{t^*-1}$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
    Else if $t = t^*-1$ **and** $m_{\text{out}} \neq m_{t^*-1}$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 40: Program P_{int}

Hybrid H'_{int} This hybrid is similar to H_{int}, except that the challenger also computes $b^* = M_b(x)$ and outputs an obfuscation of $P'_{int} = P'_{int}\{t^*, K_E, K_A, K_B, m_{t^*-1}, b^*\}$ (defined in Figure 41). This program is identical to P_{int}, except for inputs corresponding to $t = t^*$. At $t = t^*$, the program verifies the validity of signature, and then outputs b^* (which it has hardwired). It does not decrypt the ciphertexts.

[5]There is a slight difference between the approach here and the one in the proof of Theorem 6.1. There, we allowed the final program to output 'B' type signatures. Here, in order to remove the 'B' signatures completely, we use additional t^* hybrids

Program P'_{int}

Constants: Turing machine $M = \langle Q, \Sigma_{\rm tape}, \delta, q_0, q_{\rm acc}, q_{\rm rej}\rangle$, time bound T, halt-time $t^* \in [T]$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, message m_{t^*-1}, bit b^*.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\rm in}$, accumulator value $w_{\rm in} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature $\sigma_{\rm in}$, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\rm in} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\rm out} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw}), \mathsf{pos}_{\rm in}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
3. Let $r_{S,A} = F(K_A, t-1)$, $r_{S,B} = F(K_B, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$ and $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,B})$.
4. Let $m_{\rm in} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in})$.
5. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 0$ output $\perp$.
6. If $t = t^*$ output b^*.
7. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\rm in})$.
8. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\rm in})$.
9. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
10. If $\mathsf{st}_{\rm out} = q_{\rm rej}$ output 0, else if $\mathsf{st}_{\rm out} = q_{\rm acc}$ output 1.
11. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
12. Compute $w_{\rm out} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm out}, t), \mathsf{pos}_{\rm in}, aux)$. If $w_{\rm out} = Reject$, output $\perp$.
13. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in}))$.
14. Let $r'_{S,A} = F(K_A, t)$, $r'_{S,B} = F(K_B, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$, $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
15. Let $m_{\rm out} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\rm out}, w_{\rm out}, \mathsf{pos}_{\rm out})$.
    If $t = t^* - 1$ **and** $m_{\rm out} = m_{t^*-1}$, $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\rm out})$.
    Else if $t = t^* - 1$ **and** $m_{\rm out} \neq m_{t^*-1}$, $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\rm out})$.
    Else, $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\rm out})$.
16. Output $\mathsf{pos}_{\rm in}, \mathsf{ct}_{\mathsf{sym},\rm out}, \mathsf{ct}_{\mathsf{ct},\rm out}, w_{\rm out}, v_{\mathsf{out}}, \sigma_{\rm out}$.

Figure 41: Program P'_{int}

Hybrid H_{abort} In this hybrid, the challenger outputs an obfuscation of $P_{abort}\{t^*, K_A, K_E, b^*\}$ (defined in Figure 42). This program is similar to P'_{int}, except that it does not output 'B' type signatures.

Let $\mathsf{Adv}^{int}_{\mathcal{A}}$, $\mathsf{Adv}^{'int}_{\mathcal{A}}$, $\mathsf{Adv}^{abort}_{\mathcal{A}}$ be the advantages of an adversary $\mathcal{A}$ in H_{int}, H'_{int} and H_{abort} respectively. Recall $\mathsf{Adv}^0_{\mathcal{A}}$ and $\mathsf{Adv}^1_{\mathcal{A}}$ denote $\mathcal{A}$'s advantage in Hyb_0 and Hyb_1 respectively.

Lemma B.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^{int}_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this lemma is very similar to the proof of Theorem 6.1. Therefore, in this section, we will give outline of the proof, consisting of the outer hybrids, and refer to proof of Theorem 6.1. We will first define intermediate hybrids H_0, H_1 and $H_{2,j}, H'_{2,j}$ for $0 \leq j < t^*$.

Hybrid H_0 The challenger outputs $P_0 = \mathsf{Prog}\{t^*, K_E, K_A\}$.

Hybrid H_1 The challenger outputs $P_1 = P_1\{t^*, K_E, K_A, K_B\}$ (defined in Figure 43). This is similar to $\mathsf{Prog\text{-}1}$ defined in Figure 18. This program has PRF key K_B hardwired and accepts both 'A' and 'B' type signatures for $t < t^*$. If the incoming signature is of type α, then so is the outgoing signature.

that 'undo' the step from Prog to P_{abort}.

Program P_{abort}

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \in [T]$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_A \in \mathcal{K}$, bit b^*.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
3. Let $r_{S,A} = F(K_A, t-1)$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
4. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$.
5. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
6. If $t = t^*$ output b^*.
7. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
8. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
9. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
10. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0, else if $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
11. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
12. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
13. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
14. Let $r'_{S,A} = F(K_A, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
15. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
16. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 42: Program P_{abort}

Next, we define $2t^*$ intermediate circuits - $P_{2,j}, P'_{2,j}$ for $0 \leq j \leq t^* - 1$. These programs are analogous to $\mathsf{Prog\text{-}2\text{-}}i$ and $\mathsf{Prog'\text{-}2\text{-}}i$ in the proof of Theorem 6.1.

Hybrid $H_{2,j}$ In this hybrid, the challenger outputs an obfuscation of $P_{2,j} = P_{2,j}\{j, t^*, K_E, K_A, K_B, m_j\}$. This circuit, defined in Figure 44, accepts 'B' type signatures only for inputs corresponding to $j+1 \leq t \leq t^* - 1$. It also has the correct output message for step j - m_j hardwired. If an input has $j+1 \leq t \leq t^* - 1$, then the output signature, if any, is of the same type as the incoming signature. If $t = j$, the program outputs an 'A' type signature if $m_{\text{out}} = m_j$, else it outputs a 'B' type signature.

Hybrid $H'_{2,j}$ In this hybrid, the challenger outputs an obfuscation of $P'_{2,j} = P'_{2,j}\{j, t^*, K_E, K_A, K_B, m_j\}$. This circuit, defined in Figure 45, accepts 'B' type signatures only for inputs corresponding to $j+2 \leq t \leq t^* - 1$. It also has the correct input message for step $j+1$ - m_j hardwired. If $t = j+1$ and $m_{\text{in}} = m_j$ it outputs an 'A' type signature, else it outputs a 'B' type signature. If an input has $j+2 \leq t \leq t^* - 1$, then the output signature, if any, is of the same type as the incoming signature.

Analysis

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 5.1, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

The proof of this claim is similar to the proof of Lemma 6.1.

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

Note that P_1 and $P_{2,0}$ have identical functionality.

P_1

Constants: i, Turing machine $M = \langle Q, \Sigma_{\rm tape}, \delta, q_0, q_{\rm acc}, q_{\rm rej}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, output b^*.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\rm in}$, accumulator value $w_{\rm in} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature $\sigma_{\rm in}$, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\rm in} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\rm out} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw}), \mathsf{pos}_{\rm in}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. Let $F(K_B, t-1) = r_{S,B}$. Compute $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,B})$.
7. Let $F(K_B, t) = r'_{S,B}$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
8. Let $m_{\rm in} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in})$ and $\alpha =$'-'.
   If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 1$ set $\alpha =$'A'.
   If $\alpha =$'-' **and** $t \geq t^*$ output $\perp$.
   If $\alpha =$ '-' **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\rm in}, \sigma_{\rm in}) = 1$ set $\alpha =$'B'.
   If $\alpha =$ '-' output $\perp$.
9. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\rm in})$.
10. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\rm in})$.
11. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
12. If $\mathsf{st}_{\rm out} = q_{\rm rej}$ output 0.
13. If $\mathsf{st}_{\rm out} = q_{\rm acc}$ output 1.
14. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
15. Compute $w_{\rm out} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm out}, t), \mathsf{pos}_{\rm in}, aux)$. If $w_{\rm out} = Reject$, output $\perp$.
16. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in}))$.
17. Let $m_{\rm out} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\rm out}, w_{\rm out}, \mathsf{pos}_{\rm out})$ and $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\rm out})$.
18. Output $\mathsf{pos}_{\rm in}, \mathsf{ct}_{\mathsf{sym},\rm out}, \mathsf{ct}_{\mathsf{ct},\rm out}, w_{\rm out}, v_{\mathsf{out}}, \sigma_{\rm out}$.

Figure 43: P_1

Claim B.3. Let $0 \leq j \leq t^* - 1$. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF and $\mathcal{S}$ is a splittable signature scheme satisfying definitions 5.1, 5.2, 5.3 and 5.4, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{2,j}_{\mathcal{A}} - \mathsf{Adv}'^{2,j}_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

The proof of this claim is similar to the proof of Lemma 6.2.

Claim B.4. Let $0 \leq j \leq t^* - 2$. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, Itr is an iterator satisfying indistinguishability of Setup (Definition 3.1) and is enforcing (Definition 3.2), and Acc is an accumulator satisfying indistinguishability of Read/Write Setup (Definitions 4.1 and 4.2) and is Read/Write enforcing (Definitions 4.3 and 4.4), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}'^{2,j}_{\mathcal{A}} - \mathsf{Adv}^{2,j+1}_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$. indistinguishable.

The proof of this claim is similar to the proof of Lemma 6.3.

Claim B.5. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{2,t^*-1}_{\mathcal{A}} - \mathsf{Adv}^{int}_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Note that P_{2,t^*-1} and P_{int} are functionally identical circuits. This completes the proof of our lemma. ∎

$P_{2,j}$

Constants: j, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, output b^*, message m_j.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
3. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
4. Let $F(K_A, t) = r'_{S,A}$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
5. Let $F(K_B, t-1) = r_{S,B}$. Compute $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,B})$.
6. Let $F(K_B, t) = r'_{S,B}$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
7. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$ and $\alpha =$'-'.
   If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'A'.
   If $\alpha =$'-' **and** ($t \geq t^*$ **or** $t \leq j$) output $\perp$.
   If $\alpha =$ '-' **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 1$ set $\alpha =$'B'.
   If $\alpha =$ '-' output $\perp$.
8. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
9. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
10. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
11. If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
12. If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
13. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
14. Compute $w_{\text{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
15. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
16. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = j$ **and** $m_{\text{out}} = m_j$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\text{out}})$.
    Else if $t = j$ **and** $m_{\text{out}} \neq m_j$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\text{out}})$.
17. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 44: $P_{2,j}$

$P'_{2,j}$

Constants: j, Turing machine $M = \langle Q, \Sigma_{\rm tape}, \delta, q_0, q_{\rm acc}, q_{\rm rej}\rangle$, time bound T, halt-time $t^* \le T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, output b^*, message m_j.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\rm in}$, accumulator value $w_{\rm in} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature $\sigma_{\rm in}$, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\rm in} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\rm out} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw}), \mathsf{pos}_{\rm in}, \pi) = 0$ **or** $\mathsf{lw} \ge t$ output $\perp$.
3. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
4. Let $F(K_A, t) = r'_{S,A}$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
5. Let $F(K_B, t-1) = r_{S,B}$. Compute $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,B})$.
6. Let $F(K_B, t) = r'_{S,B}$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
7. Let $m_{\rm in} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in})$ and $\alpha =$'-'.
   If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 1$ set $\alpha =$'A'.
   If $\alpha =$'-' **and** ($t \ge t^*$ **or** $\underline{t \le j+1}$) output $\perp$.
   If $\alpha =$ '-' **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\rm in}, \sigma_{\rm in}) = 1$ set $\alpha =$'B'.
   If $\alpha =$ '-' output $\perp$.
8. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\rm in})$.
9. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\rm in})$.
10. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
11. If $\mathsf{st}_{\rm out} = q_{\rm rej}$ output 0.
12. If $\mathsf{st}_{\rm out} = q_{\rm acc}$ output 1.
13. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
14. Compute $w_{\rm out} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm out}, t), \mathsf{pos}_{\rm in}, aux)$. If $w_{\rm out} = Reject$, output $\perp$.
15. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in}))$.
16. Let $m_{\rm out} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\rm out}, w_{\rm out}, \mathsf{pos}_{\rm out})$.
    If $t = j+1$ **and** $\underline{m_{\rm in} = m_j}$, $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\rm out})$.
    Else if $t = j+1$ **and** $\underline{m_{\rm in} \ne m_j}$, $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\rm out})$.
    Else $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\rm out})$.
17. Output $\mathsf{pos}_{\rm in}, \mathsf{ct}_{\mathsf{sym},\rm out}, \mathsf{ct}_{\mathsf{ct},\rm out}, w_{\rm out}, v_{\mathsf{out}}, \sigma_{\rm out}$.

Figure 45: $P'_{2,j}$

Lemma B.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, $|\mathsf{Adv}^{int}_{\mathcal{A}} - \mathsf{Adv}'^{int}_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. To prove this lemma, we will define a sequence of hybrid experiments and show that they are computationally indistinguishable.

Hybrid H_0 In this experiment, the challenger outputs an obfuscation of $P_0 = P_{int}\{t^*, K_E, K_A, K_B, m_{t^*-1}\}$.

Hybrid H_1 In this hybrid, the challenger first computes the constants for program P_1 as follows:

1. PRF keys K_A and K_B are punctured at t^*-1 to obtain $K_A\{t^*-1\} \leftarrow F.\mathsf{puncture}(K_A, t^*-1)$ and $K_B\{t^*-1\} \leftarrow F.\mathsf{puncture}(K_B, t^*-1)$.
2. Let $r_c = F(K_A, t^*-1)$, $(\mathrm{SK}_C, \mathrm{VK}_C, \mathrm{VK}_{C,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$, $r_D = F(K_B, t^*-1)$, $(\mathrm{SK}_D, \mathrm{VK}_D, \mathrm{VK}_{D,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_D)$.

It then outputs an obfuscation of $P_1 = P_1\{t^*, K_E, K_A\{t^*-1\}, K_B\{t^*-1\}, \mathrm{VK}_C, \mathrm{SK}_C, \mathrm{SK}_D, m_{t^*-1}\}$ (defined in 46). P_1 is identical to P_0 on inputs corresponding to $t \neq t^*-1, t^*$. For $t = t^*-1$, it uses the hardwired signing keys. For $t = t^*$, it uses the hardwired verification key.

P_1

Constants: i, Turing machine $M = \langle Q, \Sigma_{\mathrm{tape}}, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A\{t^*-1\}, K_B\{t^*-1\} \in \mathcal{K}$, output b^*, message m_{t^*-1}, VK_C, SK_C, SK_D.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\mathrm{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\mathrm{in}}$, accumulator value $w_{\mathrm{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\mathrm{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\mathrm{out}} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, (\mathsf{ct}_{\mathsf{sym},\mathrm{in}}, \mathsf{lw}), \mathsf{pos}_{\mathrm{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
3. If $t \neq t^*$, let $r_{S,A} = F.\mathsf{eval}(K_A\{t^*-1\}, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$. Else $\mathrm{VK}_A = \mathrm{VK}_C$.
4. If $t \neq t^*-1$, let $r'_{S,A} = F.\mathsf{eval}(K_A\{t^*-1\}, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
5. If $t \neq t^*-1$, $r'_{S,B} = F.\mathsf{eval}(K_B\{t^*-1\}, t)$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
6. Let $m_{\mathrm{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 0$ output $\perp$.
7. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\mathrm{in}})$.
8. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\mathrm{in}})$.
9. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
10. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{rej}}$ output 0.
11. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ output 1.
12. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\mathrm{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\mathrm{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
13. Compute $w_{\mathrm{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, (\mathsf{ct}_{\mathsf{sym},\mathrm{out}}, t), \mathsf{pos}_{\mathrm{in}}, aux)$. If $w_{\mathrm{out}} = Reject$, output $\perp$.
14. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}))$.
15. Let $m_{\mathrm{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\mathrm{out}}, w_{\mathrm{out}}, \mathsf{pos}_{\mathrm{out}})$.
16. If $t = t^*-1$ and $m_{\mathrm{out}} = m_{t^*-1}$, $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}_C, m_{\mathrm{out}})$. Else if $t = t^*-1$ and $m_{\mathrm{out}} \neq m_{t^*-1}$ $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}_D, m_{\mathrm{out}})$. Else $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\mathrm{out}})$.
17. Output $\mathsf{pos}_{\mathrm{in}}, \mathsf{ct}_{\mathsf{sym},\mathrm{out}}, \mathsf{ct}_{\mathsf{ct},\mathrm{out}}, w_{\mathrm{out}}, v_{\mathsf{out}}, \sigma_{\mathrm{out}}$.

Figure 46: P_1

Hybrid H_2 In this hybrid, r_C and r_D are chosen uniformly at random; that is, the challenger computes $(\mathrm{SK}_C, \mathrm{VK}_C) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and $(\mathrm{SK}_D, \mathrm{VK}_D) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$.

Hybrid H_3 In this hybrid, the challenger computes constrained secret/verification keys. It computes $(\sigma_{C,\mathrm{one}}, \mathrm{VK}_{C,\mathrm{one}}, \mathrm{SK}_{C,\mathrm{abo}}, \mathrm{VK}_{C,\mathrm{abo}}) \leftarrow \mathsf{Split}(\mathrm{SK}_C, m_{t^*-1})$ and $(\sigma_{D,\mathrm{one}}, \mathrm{VK}_{D,\mathrm{one}}, \mathrm{SK}_{D,\mathrm{abo}}, \mathrm{VK}_{D,\mathrm{abo}}) \leftarrow \mathsf{Split}(\mathrm{SK}_D, m_{t^*-1})$. It then outputs an obfuscation of $P_3 = P_3\{i, t^*, K_E, K_A\{t^*-1\}, K_B\{t^*-1\}, \mathrm{VK}_{C,\mathrm{one}}, \sigma_{C,\mathrm{one}}, \mathrm{SK}_{D,\mathrm{abo}}, m_{t^*-1}\}$ (defined in Figure 47). Note that $\mathrm{SK}_C, \mathrm{VK}_C$, $\mathrm{SK}_D, \mathrm{VK}_D$ are not hardwired in this program.

P_3

Constants: i, Turing machine $M = \langle Q, \Sigma_{\mathrm{tape}}, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A\{t^*-1\}, K_B\{t^*-1\} \in \mathcal{K}$, output b^*, message m_{t^*-1}, $\mathrm{VK}_{C,\mathrm{one}}$, $\sigma_{C,\mathrm{one}}$, $\mathrm{SK}_{D,\mathrm{abo}}$.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\mathrm{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\mathrm{in}}$, accumulator value $w_{\mathrm{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\mathrm{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\mathrm{out}} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, (\mathsf{ct}_{\mathsf{sym},\mathrm{in}}, \mathsf{lw}), \mathsf{pos}_{\mathrm{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
3. If $t \neq t^*$, let $r_{S,A} = F.\mathsf{eval}(K_A\{t^*-1\}, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\mathrm{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
   Else $\mathrm{VK}_A = \mathrm{VK}_{C,\mathrm{one}}$.
4. If $t \neq t^*-1$, let $r'_{S,A} = F.\mathsf{eval}(K_A\{t^*-1\}, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
5. If $t \neq t^*-1$, $r'_{S,B} = F.\mathsf{eval}(K_B\{t^*-1\}, t)$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\mathrm{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
6. Let $m_{\mathrm{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\mathrm{in}}, \sigma_{\mathrm{in}}) = 0$ output $\perp$.
7. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\mathrm{in}})$.
8. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\mathrm{in}})$.
9. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
10. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{rej}}$ output 0.
11. If $\mathsf{st}_{\mathrm{out}} = q_{\mathrm{acc}}$ output 1.
12. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\mathrm{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\mathrm{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
13. Compute $w_{\mathrm{out}} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\mathrm{in}}, (\mathsf{ct}_{\mathsf{sym},\mathrm{out}}, t), \mathsf{pos}_{\mathrm{in}}, aux)$. If $w_{\mathrm{out}} = Reject$, output $\perp$.
14. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\mathrm{in}}, w_{\mathrm{in}}, \mathsf{pos}_{\mathrm{in}}))$.
15. Let $m_{\mathrm{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\mathrm{out}}, w_{\mathrm{out}}, \mathsf{pos}_{\mathrm{out}})$.
16. If $t = t^*-1$ and $m_{\mathrm{out}} = m_{t^*-1}$, $\sigma_{\mathrm{out}} = \sigma_{C,one}$.
    Else if $t = t^*-1$ and $m_{\mathrm{out}} \neq m_{t^*-1}$ $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl\text{-}abo}(\mathrm{SK}_{D,\mathrm{abo}}, m_{\mathrm{out}})$.
    Else $\sigma_{\mathrm{out}} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\mathrm{out}})$.
17. Output $\mathsf{pos}_{\mathrm{in}}, \mathsf{ct}_{\mathsf{sym},\mathrm{out}}, \mathsf{ct}_{\mathsf{ct},\mathrm{out}}, w_{\mathrm{out}}, v_{\mathsf{out}}, \sigma_{\mathrm{out}}$.

Figure 47: P_3

Hybrid H_4 In this hybrid, the challenger chooses $\mathrm{PP}_{\mathsf{Acc}}, w_0, store_0$ using $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}$. It then uses $\mathrm{PP}_{\mathsf{Acc}}, w_0, store_0$, and proceeds as in previous experiment. It outputs an obfuscation of $P_3\{i, t^*, \mathrm{PP}_{\mathsf{Acc}}, K_E, K_A\{t^*-1\}, K_B\{t^*-1\}, \mathrm{VK}_{C,\mathrm{one}}, \sigma_{C,\mathrm{one}}, \mathrm{SK}_{D,\mathrm{abo}}, m_{t^*-1}\}$

Hybrid H_5 In this hybrid, the challenger first computes $b^* = M_b(x)$.

It then outputs an obfuscation of $P_5 = P_5\{i, t^*, \mathrm{PP}_{\mathsf{Acc}}, K_E, K_A\{t^*-1\}, K_B\{t^*-1\}, \mathrm{VK}_{C,\mathrm{one}}, \sigma_{C,\mathrm{one}}, \mathrm{SK}_{D,\mathrm{abo}}, m_{t^*-1}, b^*\}$ (defined in Figure 48). This program differs from P_3 for inputs corresponding to $t = t^*$. Instead of decrypting, computing the next state and then encrypting, the program uses the hardwired output b^*.

Hybrid H_6 In this experiment, the challenger uses normal setup for Acc (that is, $\mathsf{Setup\text{-}Acc}$) instead of $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}$.

P_5

Constants: i, Turing machine $M = \langle Q, \Sigma_{\rm tape}, \delta, q_0, q_{\rm acc}, q_{\rm rej}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A\{t^*-1\}, K_B\{t^*-1\} \in \mathcal{K}$, output b^*, message m_{t^*-1}, $\mathrm{VK}_{C,\rm one}$, $\sigma_{C,\rm one}$, $\mathrm{SK}_{D,\rm abo}$, ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\rm in}$, accumulator value $w_{\rm in} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature $\sigma_{\rm in}$, accumulator proof π, auxiliary value aux.

1. Let $\mathsf{pos}_{\rm in} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\rm out} = \mathsf{tmf}(t)$.
2. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw}), \mathsf{pos}_{\rm in}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
3. If $t \neq t^*$, let $r_{S,A} = F.\mathsf{eval}(K_A\{t^*-1\}, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$. Else $\mathrm{VK}_A = \mathrm{VK}_{C,\rm one}$.
4. If $t \neq t^*-1$, let $r'_{S,A} = F.\mathsf{eval}(K_A, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
5. If $t \neq t^*-1$, $r'_{S,B} = F.\mathsf{eval}(K_B, t)$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
6. Let $m_{\rm in} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 0$ output $\perp$.
7. If $t = t^*$ output b^*.
8. Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\rm in})$.
9. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\rm in})$.
10. Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
11. If $\mathsf{st}_{\rm out} = q_{\rm rej}$ output 0.
12. If $\mathsf{st}_{\rm out} = q_{\rm acc}$ output 1.
13. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
14. Compute $w_{\rm out} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm out}, t), \mathsf{pos}_{\rm in}, aux)$. If $w_{\rm out} = Reject$, output $\perp$.
15. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in}))$.
16. Let $m_{\rm out} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\rm out}, w_{\rm out}, \mathsf{pos}_{\rm out})$.
17. If $t = t^*-1$ and $m_{\rm out} = m_{t^*-1}$, $\sigma_{\rm out} = \sigma_{C,one}$.
    Else if $t = t^*-1$ and $m_{\rm out} \neq m_{t^*-1}$ $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl\text{-}abo}(\mathrm{SK}_{D,\rm abo}, m_{\rm out})$.
    Else $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\rm out})$.
18. Output $\mathsf{pos}_{\rm in}, \mathsf{ct}_{\mathsf{sym},\rm out}, \mathsf{ct}_{\mathsf{ct},\rm out}, w_{\rm out}, v_{\mathsf{out}}, \sigma_{\rm out}$.

Figure 48: P_5

Hybrid H_7 This hybrid is identical to H'_{int}. In this experiment, the challenger outputs an obfuscation of W'_{int}.

Analysis Let $\mathsf{Adv}^x_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in hybrid H_x.

Claim B.6. 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. The only difference between P_0 and P_1 is that P_0 uses puncturable PRF keys K_A, K_B, while P_1 uses keys $K_A\{t^*-1\}, K_B\{t^*-1\}$ punctured at t^*-1. It also has the secret key/verification key pair $(\mathrm{SK}_C, \mathrm{VK}_C)$ hardwired, which is computed using $F(K_A, t^*-1)$ and the secret key (SK_D) computed using $F(K_B, t^*-1)$. From the correctness of puncturable PRFs, it follows that the two programs have identical functionality, and therefore their obfuscations are computationally indistinguishable. ∎

Claim B.7. Assuming F is a selectively secure puncturable PRF, for any PPT $\mathcal{A}$, $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this claim is similar to the proof of Claim A.4; it follows from the selective security of puncturable PRF F. ∎

Claim B.8. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and $\mathcal{S}$ satisfies $\mathrm{VK}_{\rm one}$ indistinguishability (Definition 5.2), for any PPT $\mathcal{A}$, $|\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. In order to prove this claim, we consider an intermediate hybrid program in which only the constrained secret keys $\sigma_{C,\mathrm{one}}$ and $\mathrm{SK}_{D,\mathrm{abo}}$ are hardwired, while VK_C is hardwired as the verification key. Using the security of $i\mathcal{O}$, we can argue that the intermediate step and H_2 are computationally indistinguishable. Next, we use $\mathrm{VK}_{\mathrm{one}}$ indistinguishability to show that the intermediate step and H_3 are computationally indistinguishable. ■

Claim B.9. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT $\mathcal{A}$, $|\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this claim follows from Read Setup indistinguishability (Definition 4.1); it is similar to the proof of A.14. ■

Claim B.10. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT $\mathcal{A}$, $|\mathsf{Adv}^4_{\mathcal{A}} - \mathsf{Adv}^5_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. This proof is similar to the proof of Claim A.15. Note that since $\mathrm{PP}_{\mathsf{Acc}}$ is read enforced, and $\mathrm{VK}_{C,\mathrm{one}}$ accepts only signatures for m_{t^*-1}. As a result, if $m_{\mathrm{in}} = m_{t^*-1}$ and $\mathrm{PP}_{\mathsf{Acc}}$ is read enforced, then $\mathsf{st}_{\mathrm{out}} = \mathsf{st}_{t^*}$, which implies that the output is b^*. ■

Claim B.11. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT $\mathcal{A}$, $|\mathsf{Adv}^5_{\mathcal{A}} - \mathsf{Adv}^6_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this claim follows from Read Setup indistinguishability (Definition 4.1); it is similar to the proof of A.14. ■

Claim B.12. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF and $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{one}}$ indistinguishability (Definition 5.2), for any PPT $\mathcal{A}$, $|\mathsf{Adv}^6_{\mathcal{A}} - \mathsf{Adv}^7_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

This step is the reverse of the step from H_0 to H_3. Therefore, using similar intermediate hybrid experiments, a similar proof works here as well.

■

Lemma B.3. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, $|\mathsf{Adv}^{'int}_{\mathcal{A}} - \mathsf{Adv}^{abort}_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

The proof of this lemma is almost identical to the proof of Lemma B.1.

Lemma B.4. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure PRF and $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{rej}}$ indistinguishability (Definition 5.1), for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{abort}_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Note that at $t = t^*$, P_{abort} either outputs b^* or outputs $\perp$. This allows us to use $\mathrm{VK}_{\mathrm{rej}}$ indistinguishability to output $\perp$ for all $t > t^*$. More formally, we will define $T - t^* + 1$ hybrids $H_{abort,i}$ for $t^* \leq i \leq T$. In hybrid $H_{abort,i}$, the challenger outputs an obfuscation of $P_{abort,i}\{t^*, K_E, K_A, K_B, b^*\}$ (defined in Figure 49) which aborts if the input corresponds to $t > i$.

Clearly, P_{abort} and $P_{abort,T-1}$ are functionally identical, and $\mathsf{Prog}\text{-}1$ and P_{abort,t^*} are functionally identical. Therefore, all that remains to show is that $H_{abort,i}$ and $H_{abort,i-1}$ are computationally indistinguishable.

Claim B.13. Assuming $\mathcal{S}$ satisfies $\mathrm{VK}_{\mathrm{rej}}$ indistinguishability (Definition 5.1), $i\mathcal{O}$ is a secure indistinguishability obfuscator and F is a selectively secure pseudorandom function, for any adversary $\mathcal{A}$, $|\mathsf{Adv}^{abort,i}_{\mathcal{A}} - \mathsf{Adv}^{abort,i-1}_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Program $P_{abort,i}$

Constants: Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \in [T]$, Public parameters for accumulator $\text{PP}_{\textsf{Acc}}$, Public parameters for Iterator $\text{PP}_{\textsf{Itr}}$, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, message m_{t^*-1}, bit b^*.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\textsf{ct}_{\textsf{sym},\text{in}}, \textsf{lw})$, encrypted state $\textsf{ct}_{\textsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\textsf{Acc}}}$, Iterator value $v_{\textsf{in}}$, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > i$, output $\perp$.
2. Let $\textsf{pos}_{\text{in}} = \textsf{tmf}(t-1)$ and $\textsf{pos}_{\text{out}} = \textsf{tmf}(t)$.
3. If $\textsf{Verify-Read}(\text{PP}_{\textsf{Acc}}, w_{\text{in}}, (\textsf{ct}_{\textsf{sym},\text{in}}, \textsf{lw}), \textsf{pos}_{\text{in}}, \pi) = 0$ **or** $\textsf{lw} \geq t$ output $\perp$.
4. Let $r_{S,A} = F(K_A, t-1)$, $r_{S,B} = F(K_B, t-1)$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \textsf{Setup-Spl}(1^\lambda; r_{S,A})$ and $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \textsf{Setup-Spl}(1^\lambda; r_{S,B})$.
5. Let $m_{\text{in}} = (v_{\textsf{in}}, \textsf{ct}_{\textsf{st},\text{in}}, w_{\text{in}}, \textsf{pos}_{\text{in}})$.
6. If $\textsf{Verify-Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
7. If $t = t^*$ output b^*.
8. Let $(r_{\textsf{lw},1}, r_{\textsf{lw},2}, r_{\textsf{lw},3}) = F(K_E, \textsf{lw})$, $(\textsf{pk}_{\textsf{lw}}, \textsf{sk}_{\textsf{lw}}) = \textsf{Setup-PKE}(1^\lambda; r_{\textsf{lw},1})$, $\textsf{sym} = \textsf{Dec-PKE}(\textsf{sk}_{\textsf{lw}}, \textsf{ct}_{\textsf{sym},\text{in}})$.
9. Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\textsf{pk}_{\textsf{st}}, \textsf{sk}_{\textsf{st}}) = \textsf{Setup-PKE}(1^\lambda, r_{t-1,1})$, $\textsf{st} = \textsf{Dec-PKE}(\textsf{sk}_{\textsf{st}}, \textsf{ct}_{\textsf{st},\text{in}})$.
10. Let $(\textsf{st}', \textsf{sym}', \beta) = \delta(\textsf{st}, \textsf{sym})$.
11. If $\textsf{st}_{\text{out}} = q_{\text{rej}}$ output 0, else if $\textsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
12. Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\textsf{pk}', \textsf{sk}') = \textsf{Setup-PKE}(1^\lambda; r'_{t,1})$, $\textsf{ct}_{\textsf{sym},\text{out}} = \textsf{Enc-PKE}(\textsf{pk}', \textsf{sym}'; r_{t,2})$ and $\textsf{ct}_{\textsf{st},\text{out}} = \textsf{Enc-PKE}(\textsf{pk}', \textsf{st}'; r_{t,3})$.
13. Compute $w_{\text{out}} = \textsf{Update}(\text{PP}_{\textsf{Acc}}, w_{\text{in}}, (\textsf{ct}_{\textsf{sym},\text{out}}, t), \textsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
14. Compute $v_{\textsf{out}} = \textsf{Iterate}(\text{PP}_{\textsf{Itr}}, v_{\textsf{in}}, (\textsf{ct}_{\textsf{st},\text{in}}, w_{\text{in}}, \textsf{pos}_{\text{in}}))$.
15. Let $r'_{S,A} = F(K_A, t)$, $r'_{S,B} = F(K_B, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \textsf{Setup-Spl}(1^\lambda; r'_{S,A})$, $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \textsf{Setup-Spl}(1^\lambda; r'_{S,B})$.
16. Let $m_{\text{out}} = (v_{\textsf{out}}, \textsf{ct}_{\textsf{st},\text{out}}, w_{\text{out}}, \textsf{pos}_{\text{out}})$.
    $\sigma_{\text{out}} = \textsf{Sign-Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
17. Output $\textsf{pos}_{\text{in}}, \textsf{ct}_{\textsf{sym},\text{out}}, \textsf{ct}_{\textsf{ct},\text{out}}, w_{\text{out}}, v_{\textsf{out}}, \sigma_{\text{out}}$.

Figure 49: Program $P_{abort,i}$

The proof of this claim is similar to the proof of Lemma 6.5. ∎

B.2 Proof of Lemma 7.2

Proof Intuition The main difference between $W_{2,i} = \mathsf{Prog}\text{-}2\text{-}i\{i,\, t^*,\, K_E,\, K_A\}$ and $W'_{2,i} = \mathsf{Prog}'\text{-}2\text{-}i\{i,\, t^*,\, K_E,\, K_A,\, \mathsf{ct}_1, \mathsf{ct}_2\}$ is their behavior on inputs corresponding to $t = i - 1$. $W_{2,i}$, on an input corresponding to $t = i - 1$, checks if the signature verifies, then decrypts the two ciphertexts $\mathsf{ct}_{\mathsf{st},\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{in}}$ and outputs an encryption of the next state and symbol. $W'_{2,i}$, on the other hand, does not decrypt the incoming ciphertexts. If the signatures verify, then it sets the ciphertexts to be the 'correct ciphertexts' (which are hardwired in $W'_{2,i}$).To show that the obfuscations of $W_{2,i}$ and $W'_{2,i}$ are computationally indistinguishable, we will define two intermediate program W_{int} and W'_{int}. The following table illustrates the differences:

Input corr. to	W_{int}	W'_{int}
$t > t^*$	Output $\perp$.	Output $\perp$.
$t = t^*$	Output b^*.	Output b^*.
$i \le t < t^*$	Verify σ_{in} using 'A' verification key. Set $\mathsf{ct}_{\mathsf{sym},\text{out}}$ and $\mathsf{ct}_{\mathsf{st},\text{out}}$ to be encryptions of erase.	Verify σ_{in} using 'A' verification key. Set $\mathsf{ct}_{\mathsf{sym},\text{out}}$ and $\mathsf{ct}_{\mathsf{st},\text{out}}$ to be encryptions of erase.
$t = i - 1$	Verify σ_{in} using 'A' verification key. Decrypt ciphertexts, compute the next state/symbol, encrypt. If $m_{\text{in}} = m_{i-2}$ sign using 'A' secret key, else sign using 'B' secret key. secret key.	Verify σ_{in} using 'A' verification key. Set outgoing ciphertexts to be hardwired constants $\mathsf{ct}_1, \mathsf{ct}_2$.
$t = i - 2$	Verify σ_{in} using 'A' verification key. Decrypt ciphertexts, compute the next state/symbol, encrypt. If $m_{\text{in}} = m_{i-2}$ sign outgoing message using 'A' secret key, else sign using 'B' secret key.	Verify σ_{in} using 'A' verification key. Decrypt ciphertexts, compute the next state/symbol, encrypt. If $m_{\text{in}} = m_{i-2}$ sign outgoing message using 'A' secret key, else sign using 'B' secret key.
$t < i - 2$	Verify σ_{in} using 'A' verification key. Decrypt ciphertexts, compute the next state/symbol, encrypt, sign using 'A' secret key.	Verify σ_{in} using 'A' verification key. Decrypt ciphertexts, compute the next state/symbol, encrypt, sign using 'A' secret key.

Note that $W_{2,i}$, W_{int} differ at step $i-2$, W_{int}, W'_{int} at step $i-1$ and W'_{int}, $W'_{2,i}$ at inputs corresponding to $t = i-2$. We will first argue that $W_{2,i}$ and W_{int} are computationally indistinguishable. This proof is very similar to the proof of Theorem 6.1. Next, we will show that W_{int} and W'_{int} are indistinguishable. Intuitively, in order to distinguish between the two programs, one must send an input corresponding to $t = i - 1$ with an 'A' type signature on a message $m_{\text{in}} \neq m_{i-2}$. But since both programs output an 'A' type signature only for $m_{\text{out}} = m_{i-2}$, we can replace the verification key by a restricted one that accepts only if the signature corresponds to $m_{\text{in}} = m_{i-2}$. Then, using the enforcing properties of the accumulator and iterator, we can argue that both programs are indistinguishable. Finally, note that the argument from W'_{int} to $W'_{2,i}$ is very similar to the one from $W_{2,i}$ to W_{int}.

Proof Outline As discussed in the intuition above, we will first define the programs $W_{int} = \mathsf{Prog}\text{-}2\text{-}i_{int}\{i,\, t^*,\, K_E,\, K_A,\, K_B,\, m_{i-2}\}$ (defined in Figure 50) and $W'_{int} = \mathsf{Prog}'\text{-}2\text{-}i_{int}\{i,\, t^*,\, K_E,\, K_A,\, K_B,\, m_{i-2}, \mathsf{ct}_1, \mathsf{ct}_2\}$ (defined in 51). Both the programs have the correct message for the $(i-2)^{th}$ step - m_{i-2} hardwired, and also have a PRF key K_B for 'B' type signatures. In addition, W'_{int} also has ciphertexts ct_1 and ct_2 hardwired. These are encryptions of the state and symbol output at $(i-1)^{th}$ step, computed as described in hybrid $\mathsf{Hyb}'_{2,i}$.

W_{int}

Constants: i, Turing machine $M = \langle Q, \Sigma_{\rm tape}, \delta, q_0, q_{\rm acc}, q_{\rm rej}\rangle$, time bound T, halt-time $t^* \le T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, output b^*, message m_{i-2}.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\rm in}$, accumulator value $w_{\rm in} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature $\sigma_{\rm in}$, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\rm in} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\rm out} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw}), \mathsf{pos}_{\rm in}, \pi) = 0$ **or** $\mathsf{lw} \ge t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. Let $F(K_B, t) = r'_{S,B}$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
7. Let $m_{\rm in} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 0$ output $\perp$.
8. If $t = t^*$, output b^*.
9. If $i \le t < t^*$
   (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$.
   $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
10. Else
    (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\rm in})$.
    (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\rm in})$.
    (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
    (d) If $\mathsf{st}_{\rm out} = q_{\rm rej}$ output 0.
    (e) If $\mathsf{st}_{\rm out} = q_{\rm acc}$ output 1.
    (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
11. Compute $w_{\rm out} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm out}, t), \mathsf{pos}_{\rm in}, aux)$. If $w_{\rm out} = Reject$, output $\perp$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in}))$.
13. Let $m_{\rm out} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\rm out}, w_{\rm out}, \mathsf{pos}_{\rm out})$.
    If $t = i-2$ **and** $m_{\rm out} = m_{i-2}$, $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\rm out})$.
    Else if $t = i-2$ **and** $m_{\rm out} = m_{i-2}$, $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_B, m_{\rm out})$.
    Else $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\rm out})$.
14. Output $\mathsf{pos}_{\rm in}, \mathsf{ct}_{\mathsf{sym},\rm out}, \mathsf{ct}_{\mathsf{ct},\rm out}, w_{\rm out}, v_{\mathsf{out}}, \sigma_{\rm out}$.

Figure 50: W_{int}

W'_{int}

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, output b^*, message m_{i-2}, ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. Let $F(K_B, t) = r'_{S,B}$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
7. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
8. If $t = t^*$, output b^*.
9. If $i \leq t < t^*$
   (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$.
   $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
10. Else if $t = i-1$, set $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{ct}_1$, $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{ct}_2$.
11. Else
    (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
    (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
    (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
    (d) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
    (e) If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
    (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
12. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
13. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
14. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = i-2$ **and** $m_{\text{out}} = m_{i-2}$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
    Else if $t = i-2$ **and** $m_{\text{out}} = m_{i-2}$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 51: W'_{int}

Let H_{int} be a hybrid experiment in which the challenger outputs an obfuscation of W_{int}, along with other elements of the encoding. Similarly, let H'_{int} be the hybrid experiment in which the challenger outputs W'_{int}. For any PPT adversary $\mathcal{A}$, let $\mathsf{Adv}^{2,i}_{\mathcal{A}}$, $\mathsf{Adv}^{int}_{\mathcal{A}}$, $\mathsf{Adv}'^{int}_{\mathcal{A}}$, $\mathsf{Adv}'^{2,i+1}_{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in $\mathsf{Hyb}_{2,i}$, H_{int}, H'_{int} and $\mathsf{Hyb}'_{2,i}$ respectively.

Lemma B.5. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, $|\mathsf{Adv}^{2,i}_{\mathcal{A}} - \mathsf{Adv}^{int}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. The proof of this lemma is along the same lines as the proof of Lemma B.1. We will define similar hybrid experiments here.

Hybrid H_0 The challenger outputs $P_0 = \mathsf{Prog\text{-}2\text{-}}i\{i,\, t^*,\, K_E,\, K_A\}$.

Hybrid H_1 The challenger outputs $P_1 = P_1\{i,\, t^*,\, K_E,\, K_A, K_B\}$. This is similar to $\mathsf{Prog\text{-}1}$ defined in Figure 18. This program has PRF key K_B hardwired and accepts both 'A' and 'B' type signatures for $t \leq i-2$. If the incoming signature is of type α, then so is the outgoing signature. It is defined in Figure 52.

Next, we define $2(i-1)$ intermediate circuits - $P_{2,j}, P'_{2,j}$ for $0 \leq j \leq i-2$. These programs are analogous to $\mathsf{Prog\text{-}2\text{-}}i$ and $\mathsf{Prog}'\mathsf{\text{-}2\text{-}}i$ in the proof of Theorem 6.1.

Hybrid $H_{2,j}$ In this hybrid, the challenger outputs an obfuscation of $P_{2,j} = P_{2,j}\{i, j, t^*, K_E, K_A, K_B, m_j\}$. This circuit, defined in Figure 53, accepts 'B' type signatures only for inputs corresponding to $j+1 \leq t \leq i-2$. It also has the correct output message for step j - m_j hardwired. If an input has $j+1 \leq t \leq i-2$, then the output signature, if any, is of the same type as the incoming signature.

Hybrid $H'_{2,j}$ In this hybrid, the challenger outputs an obfuscation of $P'_{2,j} = P'_{2,j}\{i, j, t^*, K_E, K_A, K_B, m_j\}$. This circuit, defined in Figure 54, accepts 'B' type signatures only for inputs corresponding to $j+2 \leq t \leq i-2$. It also has the correct input message for step $j+1$ - m_j hardwired. If $t = j+1$ and $m_{\text{in}} = m_j$ it outputs an 'A' type signature, else it outputs a 'B' type signature. If an input has $j+2 \leq t \leq i-2$, then the output signature, if any, is of the same type as the incoming signature.

Analysis Let $\mathsf{Adv}^{x}_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in hybrid H_x.

Claim B.14. 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 5.1, $\mathsf{Adv}^{0}_{\mathcal{A}} - \mathsf{Adv}^{1}_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. The proof of this claim is similar to the proof of Lemma 6.1. ∎

Claim B.15. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, $\mathsf{Adv}^{1}_{\mathcal{A}} - \mathsf{Adv}^{2,0}_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. Note that P_1 and $P_{2,0}$ have identical functionality. ∎

Claim B.16. Let $0 \leq j \leq i-2$. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF and $\mathcal{S}$ is a splittable signature scheme satisfying definitions 5.1, 5.2, 5.3 and 5.4, $\mathsf{Adv}^{2,j}_{\mathcal{A}} - \mathsf{Adv}'^{2,j}_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. The proof of this claim is similar to the proof of Lemma 6.2. ∎

P_1

Constants: i, Turing machine $M = \langle Q, \Sigma_{\rm tape}, \delta, q_0, q_{\rm acc}, q_{\rm rej}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, output b^*.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\rm in}$, accumulator value $w_{\rm in} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value $v_{\sf in}$, signature $\sigma_{\rm in}$, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\rm in} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\rm out} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw}), \mathsf{pos}_{\rm in}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. Let $F(K_B, t-1) = r_{S,B}$. Compute $(\mathrm{SK}_B, \mathrm{VK}_B, \mathrm{VK}_{B,\rm rej}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,B})$.
7. Let $F(K_B, t) = r'_{S,B}$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
8. Let $m_{\rm in} = (v_{\sf in}, \mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in})$ and $\alpha =$ 'A'.
   If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 0$ **and** $t \geq i-1$ output $\perp$.
   Else if $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 0$ set $\alpha =$ 'B'.
   If $\alpha =$ 'B' **and** $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_B, m_{\rm in}, \sigma_{\rm in}) = 0$ output $\perp$.
9. If $t = t^*$, output b^*.
10. If $i \leq t < t^*$
    (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$.
    $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
11. Else
    (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\rm in})$.
    (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\rm in})$.
    (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
    (d) If $\mathsf{st}_{\rm out} = q_{\rm rej}$ output 0.
    (e) If $\mathsf{st}_{\rm out} = q_{\rm acc}$ output 1.
    (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
12. Compute $w_{\rm out} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm out}, t), \mathsf{pos}_{\rm in}, aux)$. If $w_{\rm out} = Reject$, output $\perp$.
13. Compute $v_{\sf out} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\sf in}, (\mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in}))$.
14. Let $m_{\rm out} = (v_{\sf out}, \mathsf{ct}_{\mathsf{st},\rm out}, w_{\rm out}, \mathsf{pos}_{\rm out})$ and $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_\alpha, m_{\rm out})$.
15. Output $\mathsf{pos}_{\rm in}, \mathsf{ct}_{\mathsf{sym},\rm out}, \mathsf{ct}_{\mathsf{ct},\rm out}, w_{\rm out}, v_{\sf out}, \sigma_{\rm out}$.

Figure 52: P_1

Claim B.17. Let $0 \leq j \leq i-3$. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, Itr is an iterator satisfying indistinguishability of Setup (Definition 3.1) and is enforcing (Definition 3.2), and Acc is an accumulator satisfying indistinguishability of Read/Write Setup (Definitions 4.1 and 4.2) and is Read/Write enforcing (Definitions 4.3 and 4.4), $\mathsf{Adv}^{\prime 2,j}_{\mathcal{A}} - \mathsf{Adv}^{2,j+1}_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this claim is similar to the proof of Lemma 6.3. ∎

Claim B.18. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, $\mathsf{Adv}^{2,i-2}_{\mathcal{A}} - \mathsf{Adv}^{int}_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$.

Proof. Note that $P_{2,i-2}$ and W_{int} are functionally identical circuits. ∎

∎

$P_{2,j}$

Constants: i, j, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, output b^*, message m_j.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify}\text{-}\mathsf{Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup}\text{-}\mathsf{Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup}\text{-}\mathsf{Spl}(1^\lambda; r'_{S,A})$.
6. Let $F(K_B, t-1) = r_{S,B}$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup}\text{-}\mathsf{Spl}(1^\lambda; r_{S,B})$.
7. Let $F(K_B, t) = r'_{S,B}$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup}\text{-}\mathsf{Spl}(1^\lambda; r'_{S,B})$.
8. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$ and $\alpha =$'A'.
   If $\mathsf{Verify}\text{-}\mathsf{Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ **and** $(t \leq j$ **or** $t \geq i-1)$ output $\perp$.
   Else if $\mathsf{Verify}\text{-}\mathsf{Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ set $\alpha =$'B'.
   If $\mathsf{Verify}\text{-}\mathsf{Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
9. If $t = t^*$, output b^*.
10. If $i \leq t < t^*$
    (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup}\text{-}\mathsf{PKE}(1^\lambda; r'_{t,1})$.
    $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc}\text{-}\mathsf{PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc}\text{-}\mathsf{PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
11. Else
    (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup}\text{-}\mathsf{PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec}\text{-}\mathsf{PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
    (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup}\text{-}\mathsf{PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec}\text{-}\mathsf{PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
    (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
    (d) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
    (e) If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
    (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup}\text{-}\mathsf{PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc}\text{-}\mathsf{PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc}\text{-}\mathsf{PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
12. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
13. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
14. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = j$ **and** $m_{\text{out}} = m_j$, $\sigma_{\text{out}} = \mathsf{Sign}\text{-}\mathsf{Spl}(\text{SK}'_A, m_{\text{out}})$.
    Else if $t = j$ **and** $m_{\text{out}} \neq m_j$, $\sigma_{\text{out}} = \mathsf{Sign}\text{-}\mathsf{Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign}\text{-}\mathsf{Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 53: $P_{2,j}$

Lemma B.6. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, $|\mathsf{Adv}^{int}_{\mathcal{A}} - \mathsf{Adv}'^{int}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. The proof of this lemma is similar to the proof of Lemma B.2. To prove this lemma, we will define a sequence of hybrid experiments and show that they are computationally indistinguishable.

Hybrid H_0 In this experiment, the challenger outputs an obfuscation of $P_0 = W_{int} = \mathsf{Prog}\text{-}2\text{-}i\{i, t^*, K_E, K_A, K_B, m_{i-2}\}$.

$P'_{2,j}$

Constants: i, j, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_A, K_B \in \mathcal{K}$, output b^*, message m_j.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. Let $F(K_B, t-1) = r_{S,B}$. Compute $(\text{SK}_B, \text{VK}_B, \text{VK}_{B,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,B})$.
7. Let $F(K_B, t) = r'_{S,B}$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
8. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$ and $\alpha =$'A'.
   If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ **and** $(t \leq j+1$ **or** $t \geq i-1)$ output $\perp$.
   Else if $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ set $\alpha =$'B'.
   If $\mathsf{Verify\text{-}Spl}(\text{VK}_B, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
9. If $t = t^*$, output b^*.
10. If $i \leq t < t^*$
    (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$.
    $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
11. Else
    (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
    (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
    (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
    (d) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
    (e) If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
    (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
12. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
13. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
14. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
    If $t = j+1$ **and** $m_{\text{in}} = m_j$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
    Else if $t = j+1$ **and** $m_{\text{in}} \neq m_j$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_B, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_\alpha, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 54: $P'_{2,j}$

Hybrid H_1 In this hybrid, the challenger first computes the constants for program P_1 as follows:

1. PRF keys K_A and K_B are punctured at $i-2$ to obtain $K_A\{i-2\} \leftarrow F.\mathsf{puncture}(K_A, i-2)$ and $K_B\{i-2\} \leftarrow F.\mathsf{puncture}(K_B, i-2)$.
2. Let $r_c = F(K_A, i{-}2)$, $(\text{SK}_C, \text{VK}_C, \text{VK}_{C,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_C)$, $r_D = F(K_B, i{-}2)$, $(\text{SK}_D, \text{VK}_D, \text{VK}_{D,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_D)$.

It then outputs an obfuscation of $P_1 = P_1\{i, t^*, K_E, K_A\{i{-}2\}, K_B\{i{-}2\}, \text{VK}_{C,\text{one}}, \text{SK}_{C,\text{one}}, \text{SK}_{D,\text{abo}}, m_{i-2}\}$ (defined in 55). P_1 is identical to P_0 on inputs corresponding to $t \neq i-1, i-2$. However, for $i-2$, its output signature is computed using either SK_C or SK_D. For inputs corresponding to $t = i-1$, it uses VK_C for the verification.

P_1

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_A\{i-2\}, K_B\{i-2\} \in \mathcal{K}$, output b^*, message m_{i-2}, VK_C, σ_C, SK_D.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. If $t \neq i-1$, let $r_{S,A} = F.\mathsf{eval}(K_A\{i-2\}, t-1)$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
   Else $\text{VK}_A = \text{VK}_{C,\text{one}}$.
5. If $t \neq i-2$, let $r'_{S,A} = F.\mathsf{eval}(K_A\{i-2\}, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. If $t \neq i-2$, $r'_{S,B} = F.\mathsf{eval}(K_B\{i-2\}, t)$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
7. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
8. If $t = t^*$, output b^*.
9. If $i \leq t < t^*$
   (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$.
   $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
10. Else
   (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
   (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
   (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
   (d) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
   (e) If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
   (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
14. If $t = i-2$ and $m_{\text{out}} = m_{i-2}$, $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}_C, m_{\text{out}})$.
   Else if $t = i-2$ and $m_{\text{out}} \neq m_{i-2}$ $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}_D, m_{\text{out}})$.
   Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 55: P_1

Hybrid H_2 In this hybrid, r_C and r_D are chosen uniformly at random; that is, the challenger computes $(\text{SK}_C, \text{VK}_C) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$ and $(\text{SK}_D, \text{VK}_D) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda)$.

Hybrid H_3 In this hybrid, the challenger computes constrained secret/verification keys. It computes $(\sigma_{C,\text{one}}, \text{VK}_{C,\text{one}}, \text{SK}_{C,\text{abo}}, \text{VK}_{C,\text{abo}}) \leftarrow \mathsf{Split}(\text{SK}_C, m_{i-2})$ and $(\sigma_{D,\text{one}}, \text{VK}_{D,\text{one}}, \text{SK}_{D,\text{abo}}, \text{VK}_{D,\text{abo}}) \leftarrow \mathsf{Split}(\text{SK}_D, m_{i-2})$. It then outputs an obfuscation of $P_3 = \{i, t^*, K_E, K_A\{i-2\}, K_B\{i-2\}, \text{VK}_{C,\text{one}}, \sigma_{C,\text{one}}, \text{SK}_{D,\text{abo}}, m_{i-2}\}$ (defined in Figure 56). Note that SK_C, VK_C, SK_D, VK_D are not hardwired in this program.

Hybrid H_4 In this hybrid, the challenger chooses $\text{PP}_{\mathsf{Acc}}, w_0, store_0$ using $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}$. It then uses $\text{PP}_{\mathsf{Acc}}, w_0, store_0$, and proceeds as in previous experiment. It outputs an obfuscation of $P_1\{i, t^*, \text{PP}_{\mathsf{Acc}}$, K_E, $K_A\{i-2\}$, $K_B\{i-2\}$, $\text{VK}_{C,\text{one}}$, $\sigma_{C,\text{one}}$, $\text{SK}_{D,\text{abo}}$, $m_{i-2}\}$

P_3

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E, K_A\{i-2\}, K_B\{i-2\} \in \mathcal{K}$, output b^*, message m_{i-2}, $\text{VK}_{C,\text{one}}$, $\sigma_{C,\text{one}}$, $\text{SK}_{D,\text{abo}}$.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}}$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. If $t \neq i-1$, let $r_{S,A} = F.\mathsf{eval}(K_A\{i-2\}, t-1)$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
   Else $\text{VK}_A = \text{VK}_{C,\text{one}}$.
5. If $t \neq i-2$, let $r'_{S,A} = F.\mathsf{eval}(K_A\{i-2\}, t)$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. If $t \neq i-2$, $r'_{S,B} = F.\mathsf{eval}(K_B\{i-2\}, t)$. Compute $(\text{SK}'_B, \text{VK}'_B, \text{VK}'_{B,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
7. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
8. If $t = t^*$, output b^*.
9. If $i \leq t < t^*$
   (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$.
   $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
10. Else
    (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
    (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
    (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
    (d) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
    (e) If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
    (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, t), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = Reject$, output $\perp$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$.
14. If $t = i-2$ and $m_{\text{out}} = m_{i-2}$, $\sigma_{\text{out}} = \sigma_{C,one}$.
    Else if $t = i-2$ and $m_{\text{out}} \neq m_{i-2}$ $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl\text{-}abo}(\text{SK}_{D,\text{abo}}, m_{\text{out}})$.
    Else $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
15. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 56: P_3

Hybrid H_5 In this hybrid, the challenger first computes ciphertexts ct_1 and ct_2 as described in $\mathsf{Hyb}'_{2,i}$.

It then outputs an obfuscation of $P_5 = P_5\{i, t^*, \text{PP}_{\mathsf{Acc}}, K_E, K_A\{i-2\}, K_B\{i-2\}, \text{VK}_{C,\text{one}}, \sigma_{C,\text{one}}, \text{SK}_{D,\text{abo}}, m_{i-2}, \mathsf{ct}_1, \mathsf{ct}_2\}$ (defined in Figure 57). This program differs from P_3 for inputs corresponding to $t = i-1$. Instead of decrypting, computing the next state and then encrypting, the program uses the hardwired ciphertexts.

Hybrid H_6 In this experiment, the challenger uses normal setup for Acc (that is, $\mathsf{Setup\text{-}Acc}$) instead of $\mathsf{Setup\text{-}Acc\text{-}Enforce\text{-}Read}$.

Hybrid H_7 In this experiment, the challenger outputs an obfuscation of W'_{int}.

Analysis Let $\mathsf{Adv}^x_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in hybrid H_x.

P_5

Constants: i, Turing machine $M = \langle Q, \Sigma_{\rm tape}, \delta, q_0, q_{\rm acc}, q_{\rm rej}\rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator $\mathrm{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathrm{PP}_{\mathsf{Itr}}$, Puncturable PRF keys $K_E, K_A\{i-2\}, K_B\{i-2\} \in \mathcal{K}$, output b^*, message m_{i-2}, $\mathrm{VK}_{C,\rm one}$, $\sigma_{C,\rm one}$, $\mathrm{SK}_{D,\rm abo}$, ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$.

Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\rm in}$, accumulator value $w_{\rm in} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature $\sigma_{\rm in}$, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\rm in} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\rm out} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm in}, \mathsf{lw}), \mathsf{pos}_{\rm in}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. If $t \neq i-1$, let $r_{S,A} = F.\mathsf{eval}(K_A\{i-2\}, t-1)$. Compute $(\mathrm{SK}_A, \mathrm{VK}_A, \mathrm{VK}_{A,\rm rej}) = \mathsf{Sign\text{-}Spl}(1^\lambda; r_{S,A})$. Else $\mathrm{VK}_A = \mathrm{VK}_{C,\rm one}$.
5. If $t \neq i-2$, let $r'_{S,A} = F.\mathsf{eval}(K_A, t)$. Compute $(\mathrm{SK}'_A, \mathrm{VK}'_A, \mathrm{VK}'_{A,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. If $t \neq i-2$, $r'_{S,B} = F.\mathsf{eval}(K_B, t)$. Compute $(\mathrm{SK}'_B, \mathrm{VK}'_B, \mathrm{VK}'_{B,\rm rej}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,B})$.
7. Let $m_{\rm in} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in})$. If $\mathsf{Verify\text{-}Spl}(\mathrm{VK}_A, m_{\rm in}, \sigma_{\rm in}) = 0$ output $\perp$.
8. If $t = t^*$, output b^*.
9. If $i \leq t < t^*$
   (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$. $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
10. Else if $t = i-1$ set $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{ct}_1$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{ct}_2$.
11. Else
    (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F(K_E, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\rm in})$.
    (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F(K_E, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\rm in})$.
    (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
    (d) If $\mathsf{st}_{\rm out} = q_{\rm rej}$ output 0.
    (e) If $\mathsf{st}_{\rm out} = q_{\rm acc}$ output 1.
    (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F(K_E, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\rm out} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
12. Compute $w_{\rm out} = \mathsf{Update}(\mathrm{PP}_{\mathsf{Acc}}, w_{\rm in}, (\mathsf{ct}_{\mathsf{sym},\rm out}, t), \mathsf{pos}_{\rm in}, aux)$. If $w_{\rm out} = Reject$, output $\perp$.
13. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathrm{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\rm in}, w_{\rm in}, \mathsf{pos}_{\rm in}))$.
14. Let $m_{\rm out} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\rm out}, w_{\rm out}, \mathsf{pos}_{\rm out})$.
15. If $t = i-2$ and $m_{\rm out} = m_{i-2}$, $\sigma_{\rm out} = \sigma_{C,one}$.
    Else if $t = i-2$ and $m_{\rm out} \neq m_{i-2}$ $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl\text{-}abo}(\mathrm{SK}_{D,\rm abo}, m_{\rm out})$.
    Else $\sigma_{\rm out} = \mathsf{Sign\text{-}Spl}(\mathrm{SK}'_A, m_{\rm out})$.
16. Output $\mathsf{pos}_{\rm in}, \mathsf{ct}_{\mathsf{sym},\rm out}, \mathsf{ct}_{\mathsf{ct},\rm out}, w_{\rm out}, v_{\mathsf{out}}, \sigma_{\rm out}$.

Figure 57: P_5

Claim B.19. 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. In hybrid H_0, program P_0 is used, while in H_1, program P_1 is used. The only difference between the two programs is that P_1 uses punctured PRF keys $K_A\{i-2\}$ and $K_B\{i-2\}$. It also has the secret/verification keys computed using $F(K_A, i-2)$ and $F(K_B, i-2)$. As a result, using correctness of puncturable PRFs, it follows that the two programs have identical functionality. Therefore, by security of $i\mathcal{O}$, their obfuscations are computationally indistinguishable. ∎

Claim B.20. Assuming F is a selectively secure puncturable PRF, for any PPT $\mathcal{A}$, $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this claim is similar to the proof of Claim A.4; it follows from the selective security of puncturable PRF F. ∎

Claim B.21. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and $\mathcal{S}$ satisfies VK_{one} indistinguishability (Definition 5.2), for any PPT $\mathcal{A}$, $|\mathsf{Adv}^2_{\mathcal{A}} - \mathsf{Adv}^3_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. In order to prove this claim, we consider an intermediate hybrid program in which only the constrained secret keys $\sigma_{C,\text{one}}$ and $\text{SK}_{D,\text{abo}}$ are hardwired, while VK_C is hardwired as the verification key. Using the security of $i\mathcal{O}$, we can argue that the intermediate step and H_2 are computationally indistinguishable. Next, we use VK_{one} indistinguishability to show that the intermediate step and H_3 are computationally indistinguishable. ∎

Claim B.22. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT $\mathcal{A}$, $|\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. The proof of this claim follows from Read Setup indistinguishability (Definition 4.1); it is similar to the proof of A.14. ∎

Claim B.23. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT $\mathcal{A}$, $|\mathsf{Adv}^4_{\mathcal{A}} - \mathsf{Adv}^5_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This proof is similar to the proof of Claim A.15. The only additional property we require here is the correctness of decryption of $\mathcal{PKE}$. Also note that the outputs are identical because the encryption is deterministic once K_E is fixed. ∎

Claim B.24. Assuming Acc satisfies indistinguishability of Read Setup (Definition 4.1), for any PPT $\mathcal{A}$, $|\mathsf{Adv}^5_{\mathcal{A}} - \mathsf{Adv}^6_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This step is reverse of the step from H_3 to H_4, and its proof is similar to the proof of Claim A.14. ∎

Claim B.25. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF and $\mathcal{S}$ satisfies VK_{one} indistinguishability (Definition 5.2), for any PPT $\mathcal{A}$, $|\mathsf{Adv}^6_{\mathcal{A}} - \mathsf{Adv}^7_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

This step is the reverse of the step from H_0 to H_3. Therefore, using similar intermediate hybrid experiments, a similar proof works here as well. ∎

Lemma B.7. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF, Itr is an iterator satisfying Definitions 3.1 and 3.2, Acc is an accumulator satisfying Definitions 4.1, 4.2, 4.3 and 4.4, $\mathcal{S}$ is a splittable signature scheme satisfying security Definitions 5.1, 5.2, 5.3 and 5.4, $|\mathsf{Adv}^{'int}_{\mathcal{A}} - \mathsf{Adv}^{'2,i}_{\mathcal{A}}| \leq \text{negl}(\lambda)$.

The proof of this lemma is similar to the proof of Lemma B.5.

B.3 Proof of Lemma 7.3

We will first define hybrids $H_0, \ldots, H_5$, where H_0 corresponds to $\mathsf{Hyb}'_{2,i}$ and H_5 corresponds to $\mathsf{Hyb}_{2,i-1}$.

Hybrid H_0 This corresponds to $\mathsf{Hyb}'_{2,i}$.

Hybrid H_1 In this hybrid, the challenger punctures the PRF key K_E on inputs corresponding to $t = i-1$. It outputs an obfuscation of program $W_1 = \mathsf{Prog}'\text{-}2\text{-}i\text{-}1\{i, t^*, K_E\{i-1\}, K_A, \mathsf{ct}_1, \mathsf{ct}_2\}$ where Prog'-2-i-1 is defined in Figure 58. Note that the only difference between Prog'-2-i and Prog'-2-i-1 is that the latter uses a punctured PRF key $K_E\{i-1\}$ instead of K_E.

Prog'-2-i-1

Constants: i, Turing machine $M = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, time bound T, halt-time $t^* \leq T$, Public parameters for accumulator PP_{Acc}, Public parameters for Iterator PP_{Itr}, Puncturable PRF keys $K_E\{i-1\}, K_{\mathsf{st}} \in \mathcal{K}$, output b^*, ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$.
Input: Time $t \in [T]$, encrypted symbol and last-write time $(\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw})$, encrypted state $\mathsf{ct}_{\mathsf{st},\text{in}} \in Q$, accumulator value $w_{\text{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value v_{in}, signature σ_{in}, accumulator proof π, auxiliary value aux.

1. If $t > t^*$, output $\perp$.
2. Let $\mathsf{pos}_{\text{in}} = \mathsf{tmf}(t-1)$ and $\mathsf{pos}_{\text{out}} = \mathsf{tmf}(t)$.
3. If $\mathsf{Verify\text{-}Read}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{in}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, \pi) = 0$ **or** $\mathsf{lw} \geq t$ output $\perp$.
4. Let $F(K_A, t-1) = r_{S,A}$. Compute $(\text{SK}_A, \text{VK}_A, \text{VK}_{A,\text{rej}}) = \mathsf{Setup\text{-}Spl}(1^\lambda; r_{S,A})$.
5. Let $F(K_A, t) = r'_{S,A}$. Compute $(\text{SK}'_A, \text{VK}'_A, \text{VK}'_{A,\text{rej}}) \leftarrow \mathsf{Setup\text{-}Spl}(1^\lambda; r'_{S,A})$.
6. Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}})$. If $\mathsf{Verify\text{-}Spl}(\text{VK}_A, m_{\text{in}}, \sigma_{\text{in}}) = 0$ output $\perp$.
7. If $t = t^*$, output b^*.
8. If $i \leq t < t^*$
   (a) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F.\mathsf{eval}(K_E\{i-1\}, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{erase}; r_{t,3})$.
9. Else if $t = i-1$,
   (a) Set $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{ct}_1$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{ct}_2$.
10. Else
   (a) Let $(r_{\mathsf{lw},1}, r_{\mathsf{lw},2}, r_{\mathsf{lw},3}) = F.\mathsf{eval}(K_E\{i-1\}, \mathsf{lw})$, $(\mathsf{pk}_{\mathsf{lw}}, \mathsf{sk}_{\mathsf{lw}}) = \mathsf{Setup\text{-}PKE}(1^\lambda; r_{\mathsf{lw},1})$, $\mathsf{sym} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{lw}}, \mathsf{ct}_{\mathsf{sym},\text{in}})$.
   (b) Let $(r_{t-1,1}, r_{t-1,2}, r_{t-1,3}) = F.\mathsf{eval}(K_E\{i-1\}, t-1)$, $(\mathsf{pk}_{\mathsf{st}}, \mathsf{sk}_{\mathsf{st}}) = \mathsf{Setup\text{-}PKE}(1^\lambda, r_{t-1,1})$, $\mathsf{st} = \mathsf{Dec\text{-}PKE}(\mathsf{sk}_{\mathsf{st}}, \mathsf{ct}_{\mathsf{st},\text{in}})$.
   (c) Let $(\mathsf{st}', \mathsf{sym}', \beta) = \delta(\mathsf{st}, \mathsf{sym})$.
   (d) If $\mathsf{st}_{\text{out}} = q_{\text{rej}}$ output 0.
   (e) If $\mathsf{st}_{\text{out}} = q_{\text{acc}}$ output 1.
   (f) Compute $(r_{t,1}, r_{t,2}, r_{t,3}) = F.\mathsf{eval}(K_E\{i-1\}, t)$, $(\mathsf{pk}', \mathsf{sk}') = \mathsf{Setup\text{-}PKE}(1^\lambda; r'_{t,1})$, $\mathsf{ct}_{\mathsf{sym},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{sym}'; r_{t,2})$ and $\mathsf{ct}_{\mathsf{st},\text{out}} = \mathsf{Enc\text{-}PKE}(\mathsf{pk}', \mathsf{st}'; r_{t,3})$.
11. Compute $w_{\text{out}} = \mathsf{Update}(\text{PP}_{\mathsf{Acc}}, w_{\text{in}}, (\mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{lw}), \mathsf{pos}_{\text{in}}, aux)$. If $w_{\text{out}} = \mathit{Reject}$, output $\perp$.
12. Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\text{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{ct}_{\mathsf{st},\text{in}}, w_{\text{in}}, \mathsf{pos}_{\text{in}}))$.
13. Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{ct}_{\mathsf{st},\text{out}}, w_{\text{out}}, \mathsf{pos}_{\text{out}})$ and $\sigma_{\text{out}} = \mathsf{Sign\text{-}Spl}(\text{SK}'_A, m_{\text{out}})$.
14. Output $\mathsf{pos}_{\text{in}}, \mathsf{ct}_{\mathsf{sym},\text{out}}, \mathsf{ct}_{\mathsf{ct},\text{out}}, w_{\text{out}}, v_{\mathsf{out}}, \sigma_{\text{out}}$.

Figure 58: Prog'-2-i-1

Hybrid H_2 In this hybrid, the challenger computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup\text{-}PKE}(1^\lambda)$ using true randomness. Also, the ciphertexts ct_1 and ct_2 are computed using true randomness; that is, $\mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}PKE}(\mathsf{pk}, \mathsf{sym}^*)$ and $\mathsf{ct}_2 \leftarrow \mathsf{Enc\text{-}PKE}(\mathsf{pk}, \mathsf{st}^*)$.

Hybrid H_3 In this hybrid, the challenger sets $\mathsf{ct}_1 = \mathsf{Enc\text{-}PKE}(\mathsf{pk}, \mathsf{erase})$ and $\mathsf{ct}_2 = \mathsf{Enc\text{-}PKE}(\mathsf{pk}, \mathsf{erase})$.

Hybrid H_4 In this hybrid, the challenger computes the ciphertexts using pseudorandom strings generated using $F(K_E, \cdot)$. More precisely, the challenger computes $(r_{i-1,1}, r_{i-1,2}, r_{i-1,3}) = F(K_E, i-1)$, $(\mathsf{pk}, \mathsf{sk})\mathsf{Setup\text{-}PKE}(1^\lambda; r_{i-1,1})$, $\mathsf{ct}_1 = \mathsf{Enc\text{-}PKE}(\mathsf{pk}, \mathsf{erase}; r_{i-1,2})$ and $\mathsf{ct}_2 = \mathsf{Enc\text{-}PKE}(\mathsf{pk}, \mathsf{erase}; r_{i-1,3})$.

Hybrid H_5 This corresponds to $\mathsf{Hyb}_{2,i-1}$.

B.3.1 Analysis

Claim B.26. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. To prove this claim, it suffices to show that W_0 and W_1 are functionally identical. The crucial observation for this proof is the fact that $F(K_E, i-1)$ is not used anywhere in program W_0. For inputs corresponding to $t > i-1$, both programs don't use $F(K_E, i-1)$ since the programs do not decrypt for $t > i-1$. For $t = i-1$, the ciphertexts ct_1 and ct_2 are hardwired. For $t < i-1$, note that it only computes $F(K_E, \tau)$ for $\tau < i-1$. As a result, $F(K_E, \cdot)$ is not evaluated at input $i-1$, and therefore, it is safe to puncture K_E on input $i-1$ without affecting functionality. The rest follows from the correctness of puncturable PRFs. ■

Claim B.27. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. The proof of this claim is similar to the proof of Claim A.4; it follows from the selective security of puncturable PRF F. ■

Claim B.28. Assuming $\mathcal{PKE}$ is $\mathsf{IND\text{-}CPA}$ secure, 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 is not required in both hybrids H_2 and H_3. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between H_2 and H_3 with advantage ϵ. Then we can construct a PPT algorithm $\mathcal{B}$ that breaks the $\mathsf{IND\text{-}CPA}$ security of $\mathcal{PKE}$ with advantage ϵ. $\mathcal{B}$ receives the public key pk from the challenger. It interacts with $\mathcal{A}$ and computes $\mathsf{sym}^*, \mathsf{st}^*$. It sends $\boldsymbol{m_0} = (\mathsf{sym}^*, \mathsf{st}^*)$ and $\boldsymbol{m_1} = (\mathsf{erase}, \mathsf{erase})$ as the challenge message pairs, and receives a ciphertext pair$(\mathsf{ct}_1, \mathsf{ct}_2)$. $\mathcal{B}$ can now perfectly simulate H_2 or H_3 for $\mathcal{A}$, depending on whether $(\mathsf{ct}_1, \mathsf{ct}_2)$ are encryptions of $\boldsymbol{m_0}$ or $\boldsymbol{m_1}$. This completes our proof. ■

Claim B.29. Assuming F is a selectively secure puncturable PRF, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^4_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. This step is the reverse of the step from H_1 to H_2; its proof is similar to the proof of Claim A.4. ■

Claim B.30. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^4_{\mathcal{A}} - \mathsf{Adv}^5_{\mathcal{A}} \leq \text{negl}(\lambda)$.

Proof. The only difference between the programs used in the two hybrids is that one uses a punctured key $K_E\{i-1\}$, while the other uses K_E. Using the correctness of puncturable PRFs, we can argue that they are functionally identical. As a result, from the security of $i\mathcal{O}$, their obfuscations are computationally indistinguishable. ■