This technical report is available at https://eprint.iacr.org/2017/801.

Short Attribute-Based Signatures for Arbitrary Turing Machines from Standard Assumptions

Pratish Datta[1], Ratna Dutta[2], and Sourav Mukhopadhyay[2]

[1] NTT Research, Inc.
pratish.datta@ntt-research.com
[2] IIT Kharagpur
{ratna,sourav}@maths.iitkgp.ernet.in

November 30, 2020

Abstract

This paper presents the *first attribute-based signature* (ABS) scheme supporting signing policies representable by *Turing machines* (TM), based on *well-studied computational assumptions*. Our work supports *arbitrary* TMs as signing policies in the sense that the TMs can accept signing attribute strings of *unbounded* polynomial length and there is *no limit* on their running time, description size, or space complexity. Moreover, we are able to achieve *input-specific* running time for the signing algorithm. All other known expressive ABS schemes could at most support signing policies realizable by either arbitrary polynomial-size circuits or TMs having a pre-determined upper bound on the running time. Consequently, those schemes can only deal with signing attribute strings whose lengths are a priori bounded, as well as suffers from the worst-case running time problem. On a more positive note, for the *first* time in the literature, the signature size of our ABS scheme only depends on the size of the signed message and is completely *independent* of the size of the signing policy under which the signature is generated. This is a significant achievement from the point of view of communication efficiency. Our ABS construction makes use of indistinguishability obfuscation (IO) for polynomial-size circuits and certain IO-compatible cryptographic tools. Note that, all of these building blocks including IO for polynomial-size circuits are currently known to be realizable under well-studied computational assumptions.

Keywords: attribute-based signatures, Turing machines, indistinguishability obfuscation

Table of Contents

1 Introduction

In a traditional digital signature scheme, each signer possesses a secret signing key and publishes its corresponding verification key. A signature on some message issued by a certain signer is verified with respect to the public verification key of the respective signer, and hence during the verification process, the explicit signer gets identified. In other words, standard digital signatures can guarantee no privacy in the relationship between signers and claims attested by signatures due to the tight correspondence between the signing and verification keys.

Attribute-based signatures (ABS), introduced by Maji et al. [23], aims to relax such a firm relationship between signers and signatures issued by them, thereby ensuring some form of *signer privacy*. ABS comes in two flavors, namely, *key-policy* and *signature-policy*. In a key-policy ABS scheme, a setup authority holds a master signing key and publishes system public parameters. Using its master signing key, the authority can give out restricted signing keys corresponding to specific signing policies. Such a constrained signing key enables a signer to sign messages with respect to only those signing attributes which are accepted by the signing policy embedded within the signing key. The signatures are verifiable by anyone using solely the public parameters. By verifying a signature on some message with respect to some signing attributes, a verifier gets convinced that the signature is indeed generated by a signer possessing a signing key corresponding to some signing policy that accepts the signing attributes. However, the verifier cannot trace the exact signer or signing policy used to generate the signature. The signature-policy variant interchanges the roles of signing attributes and signing policies. Other than being an exciting primitive in its own right, ABS has countless interesting practical applications such as attribute-based messaging, attribute-based authentication, anonymous credential systems, trust negotiation, and leaking secrets.

A central theme of research in the field of ABS has been to expand the class of admissible signing policies in view of implementing ABS in scenarios where the correspondence between signers and signatures is more and more sophisticated. Starting with the initial work of Maji et al. [23], which supports signing policies representable by monotone span programs, the family of supported signing policies has been progressively enlarged by Okamoto and Takashima [26] to admit non-monotone span programs, by Datta et al. [9] to support arithmetic branching programs, and further by Tang et al. [30], Sakai et al. [28], Tsabary [31], as well as El Kaafarani and Katsumata [20] to realize arbitrary polynomial-size circuits. On the other hand, Bellare and Fuchsbauer [5] have put forth a versatile cryptographic primitive termed as policy-based signatures (PBS) and have exhibited a generic transformation from PBS to ABS. Their generic conversion can be used in conjunction with their proposed PBS construction to build an ABS scheme for general polynomial-size circuits as well.

While the circuit model is already powerful enough to capture arbitrary computations, an important *bottleneck* of this model is that it is *non-uniform* in nature and thus ABS schemes supporting circuit-realizable signing policies can withstand only signing attribute strings of *bounded length*, where the bound is determined during setup. Another drawback of representing signing policies as circuits is that generating a signature with respect to some signing attribute string using a signing key corresponding to certain signing policy is at least as slow as the *worst-case running time* of that policy circuit on all possible signing attribute strings. These are serious limitations not only for ABS itself, but also for all the aforementioned applications of ABS

In this paper, we aim to express signing policies in a *uniform* computational model, namely, the *Turing machine* (TM) model, which is the most natural direction to overcome the above problems. First, we would like to mention that concurrently and independently to our work, Sakai et al. [29] have developed an ABS scheme which can withstand TM-realizable signing policies under the symmetric external Diffie-Hellman (SXDH) assumption. Unfortunately however, in their ABS scheme, the size of a signature scales with the running time of the signing policy TM used to generate it on the signing attribute string with respect to which it is created. As a result, for ensuring signer privacy, their scheme should impose a universal upper bound on the running times of the signing policy TMs, and should enforce the size of the signatures to scale with that system-wide upper bound. Evidently, such a universal running-time bound in turn induces a bound on the lengths of the allowable signing attribute strings. Moreover, it implies that the signing algorithm should also have running time proportional to that universal time bound, i.e., incurs the worst-case running time in order to generate the signatures. Consequently, it is clear that their scheme actually *fails to achieve* both the advanced properties which are the sole utility of considering the richer TM model over the circuit model, namely, unbounded-length signing attribute strings and input-specific running time of the signing algorithm. Further, the failure to achieve these rich properties is in fact the result of their rather simple approach that involves giving out non-interactive zero-knowledge (NIZK) proofs for each of the evaluation steps of the signing policy TM on the signing attribute string considered

in a manner analogous to how an NIZK proof is issued for each gate of the signing policy circuit in [28]. In contrast, our goal in this paper is to devise advanced techniques to accomplish both the rich properties expected from the TM model and thereby truly expand the state of the art in the field of ABS beyond the essential barriers of the circuit model. Additionally, we aim at making the signature size as small as that of an ordinary digital signature scheme, that is, dependent only on the size of the signed message–a feature that has remained elusive despite the tremendous progress in the field of ABS so far.

1.1 Our Contribution

In this paper, we present the *first* ever key-policy ABS scheme supporting signing policies representable as *Turing machines* (TM) which can handle signing attribute strings of *unbounded polynomial length*, as well as have *arbitrary* (polynomial) running time, description size, and space complexity. Thus, our work captures the *most general* form of signing policies possible. Moreover, in our ABS scheme, generating a signing key corresponding to a signing policy takes time polynomial in the description size of that signing policy, which may be much shorter compared to the worst-case running time of that signing policy. Also, the signature generation time only depends on the time the used signing policy takes to run on the signing attribute string with respect to which the signature is being generated, rather than its worst-case running time. These features were *beyond* the reach of any other known ABS construction. On a more positive note, for the *first* time in the literature, the signature size of our ABS scheme only depends on the size of the signed message and is completely *independent* of the associated signing policy. This is a significant achievement from the point of view of communication efficiency. Further, using the technique of universal TM, our key-policy ABS construction can be readily converted into a signature-policy variant while preserving the same level of expressiveness as the key-policy version.

Our ABS construction is shown to possess *perfect signer privacy* and *existential unforgeability against selective attribute adaptive chosen message attacks* under *well-studied computational assumptions*. The construction makes use of indistinguishability obfuscation (IO) for polynomial-size circuits. Other than IO, we make use of standard digital signatures (SIG), injective pseudorandom generators (PRG), and certain additional IO-compatible cryptographic tools, namely, puncturable pseudorandom functions, somewhere statistically binding (SSB) hash functions, positional accumulators, cryptographic iterators, and splittable signatures. Among the cryptographic building blocks used in our ABS construction in addition to IO, iterators and splittable signatures are realizable using IO itself in conjunction with one-way functions, whereas all the others have very efficient instantiations based on standard number theoretic assumptions or one-way functions. Very recently, a series of exciting works [2, 22, 1, 17, 19, 13, 18] have finally provided an IO candidate based on the sub-exponential security of four well-studied computational assumptions, namely, learning with errors (LWE), learning parity with noise (LPN), existence of boolean pseudorandom generators (PRG) in NC^0, and symmetric external Diffie-Hellman (SXDH).

To achieve our result, we *extend* the techniques employed by Koppula et al. [21] for designing message-hiding encoding schemes for TMs, or by Deshpande et al. [10] for designing constrained pseudorandom functions (CPRF) for TMs secure in the selective challenge selective constraints model to withstand *adaptive* signing key queries of the adversary. We give an overview of our techniques in the next subsection.

1.2 Our Techniques

Here we explain the intuitive approach underlying our ABS scheme.

A Conventional Approach and its Drawback

The generic blueprint underlying most prominent ABS schemes so far, e.g., [24, 28, 29, 20, 5] is as follows: Each signer receives as its signing key a signature on its signing policy from the setup authority. In order to sign under a public attribute string, a signer generates a non-interactive zero-knowledge proof of knowledge (NIZK) of the signature on its signing policy that it has obtained from the setup authority and that its policy accepts the attribute string. The signature that a signer receives from the setup authority on its signing policy works as a certificate of the signer having that signing policy and prevents any third party from signing in the name of its signing policy. Also, the zero-knowledge property ensures that the proof does not leak the signature on the signer's signing policy to a verifier. It seems quite tempting to use the above generic blueprint to construct ABS scheme for TMs as well. However, this idea suffers from the following inherent limitation: If we use the above blueprint, then in order to sign a message under some signing attribute string x, the signer needs to generate an NIZK proof of the membership of x in

the language $\mathbb{L}$: $x \in \mathbb{L} \iff$ there exists some TM M belonging to the supported TM family such that M is signed by the setup authority and such that M accepts x. Now, all known NIZK proof systems are designed for NP languages. By the definition of NP, the language $\mathbb{L}$ defined above will belong to NP provided there exists a polynomial q and a verifier $\mathcal{V}$ that on input any valid instance-witness pair $(x, (M, \sigma))$ runs in time $q(|x|)$ and verifies that M accepts x and that σ is a valid signature on M. For this to hold there must exist some polynomial q' such that for any signing attribute string x and any accepting TM M belonging to the underlying TM family, the running time of M on x is bounded by $q'(|x|)$. In fact, the scheme of Sakai et al. [29], which follows this high-level construction methodology, indeed suffers from (a more stringent form of) the aforementioned limitation. This is something we want to avoid in this work. More precisely, our goal is to design ABS scheme for a TM family with no fixed polynomial bound on the running time. For such a TM family, the language $\mathbb{L}$ defined above does not belong to NP at all, so that the above generic blueprint cannot be employed.

Our Initial Idea and Its Limitations

In order to achieve signature size dependent only on the signed message, we attempt to build our key-policy ABS scheme in such a way that the signature on a message is simply a usual digital signature on it. Towards this end, we start with the following naive idea: We assign a different signing key-verification key pair of a standard digital signature (SIG) scheme to each of the possible signing attribute strings, and publish as the public parameters all the SIG verification keys associated with all the signing attribute strings, while provide the signers as their signing keys only those SIG signing keys that are associated with the signing attribute strings accepted by their signing policies. Then, in order to sign a message with respect to some signing attribute string accepted by its signing policy, a signer would sign the message using the SIG signing key associated with that signing attribute string, whereas a verifier can verify the authenticity of the signature by verifying it with respect to the SIG verification key associated with that signing attribute string. While this naive idea clearly satisfies the desired correctness and security properties of ABS, as well as fulfils our objective concerning the signature size, the immediate problem of this idea is the exponential running time of all the algorithms of the resulting ABS scheme as well as the exponential size of the public parameters and signing keys due to the potentially exponential number of possible signing attribute strings involved in the system.

Our Approach to Fix the Drawbacks of the Naive Idea

In order to solve the above issues we apply a standard derandomization technique. More precisely, we define the SIG signing key-verification key pair associated with a signing attribute string as the outcome of the setup algorithm of SIG using a pseudorandom string, which is obtained as the output of a puncturable pseudorandom function (PPRF) on input that signing attribute string using a key κ fixed during setup. Roughly speaking, a PPRF, introduced by Sahai and Waters [27], is an augmentation of a standard pseudorandom function (PRF) [14] with an additional puncturing algorithm which enables a party holding a PRF key to derive punctured keys that allow the evaluation of the PRF over all points of the input domain except one. However, given a punctured key, the PRF evaluation still remains indistinguishable from random on the input at which the key is punctured.

We design our ABS public parameters and signing keys using IO. The notion of IO [4] stipulates that the obfuscated program preserves the functionality of the original program, and the obfuscations of two functionally identical programs are computationally indistinguishable. Our public parameters and signing keys are constructed as follows: We set the public parameters of our ABS scheme to be an IO-obfuscated program $\mathcal{V}_{\text{ABS}}$, we call the verifying program, which has the PPRF key κ hardwired in it. It takes as input a signing attribute string x and performs the following steps: First, it runs the PPRF with key κ on x to generate a pseudorandom string. Next, it runs the setup algorithm of SIG using that pseudorandom string to generate and output the SIG verification key associated with x. On the other hand, the signing key corresponding to some signing policy is again an IO-obfuscated program $\mathcal{P}_{\text{ABS}}$, we call the signing program, that also has the PPRF key κ hardwired in it along with the associated signing policy. It takes as input a signing attribute string x and proceeds as follows: First it checks whether x satisfies the hardwired signing policy. If so, it generates a pseudorandom string by applying the PPRF with key κ on x, and subsequently creates and outputs the SIG signing key-verification key pair associated with x by running the setup algorithm of SIG using the generated randomness. Otherwise, if the embedded signing policy does not accepts x, then it outputs a special symbol $\perp$ indicating failure.

Thus, the setup algorithm of our modified ABS scheme simply consists of sampling a PPRF key κ and generating the obfuscated verifying program $\mathcal{V}_{\text{ABS}}$, while our signing key generation algorithm only involves generating the obfuscated signing program $\mathcal{P}_{\text{ABS}}$. To sign a message under some signing attribute string accepted by its signing policy, a signer simply runs the obfuscated signing program $\mathcal{P}_{\text{ABS}}$ contained in its signing key to obtain an SIG signing key-verification key pair, and then signs the message using the obtained SIG signing key. The ABS signature consists of the generated SIG verification key-signature pair. On the other hand, verification of an ABS signature on some message under some claimed signing attribute string requires only generating the SIG verification key associated with the claimed signing attribute string by running the obfuscated verifying program $\mathcal{V}_{\text{ABS}}$, and then checking whether the generated SIG verification key matches the one included within the ABS signature, as well as whether the SIG signature included within the ABS signature verifies under that SIG verification key. This resolves the problem of exponential running time, as well as brings the size of public parameters and signing keys down to polynomial.

The correctness of the above ABS scheme follows directly from the description of the signing and verifying programs, in conjunction with the correctness of SIG and the functionality preserving feature of IO. Let us now quickly look into the security proof of the above ABS construction. Firstly, observe that the ABS scheme clearly preserves signer privacy since the signature on some message with respect to some signing attribute string only contains the SIG verification key that has been associated with that signing attribute string while setting up the system (by sampling the PPRF key κ), together with an SIG signature on the message verifiable under that SIG verification key. In particular, the ABS signatures do not depend on the signing keys used to generate them. In order to prove selective existential unforgeability, we proceed as follows: Recall that in the selective unforgeability experiment, the adversary $\mathcal{A}$ has to commit to some signing attribute string x^*, under which it wishes to output a forgery, at the beginning of the experiment, and then is supplied with the public parameters, and is allowed to adaptively request any polynomial number of signatures and signing keys associated with signing policies that do not accept x^*. At the end, $\mathcal{A}$ outputs a forged signature on some message msg^* under x^*, and is declared to be the winner if it has not queried any signature on msg^* under x^*. To argue selective unforgeability of the above ABS construction, we first change the original unforgeability experiment into one in which we hardwire the punctured PPRF key $\kappa\{x^*\}$ punctured at x^* within the verifying program $\mathcal{V}_{\text{ABS}}$ included within the public parameters given to $\mathcal{A}$, as well as in the signing programs $\mathcal{P}_{\text{ABS}}$ included within all the signing keys provided to $\mathcal{A}$. More precisely, we modify the program $\mathcal{V}_{\text{ABS}}$ into a new program $\mathcal{V}'_{\text{ABS}}$ as follows: When run on some signing attribute string $x \neq x^*$, the program $\mathcal{V}'_{\text{ABS}}$ runs identically to $\mathcal{V}_{\text{ABS}}$, but it uses the punctured PPRF key $\kappa\{x^*\}$ in place of the full PPRF key κ. On the other hand, when run on x^*, it uses a hardwired string $\hat{r}^*_{\text{SIG}}$ as the randomness for generating the SIG verification key corresponding to x^*. We set $\hat{r}^*_{\text{SIG}}$ to be the evaluation of the PPRF with key κ on x^*. We similarly modify the signing programs $\mathcal{P}_{\text{ABS}}$ into new programs $\mathcal{P}'_{\text{ABS}}$ as follows: When run on some signing attribute string $x \neq x^*$, $\mathcal{P}'_{\text{ABS}}$ runs identically to $\mathcal{P}_{\text{ABS}}$ except that it uses the punctured PPRF key $\kappa\{x^*\}$ in place of the full PPRF key κ. On the other hand, when run on input x^*, $\mathcal{P}'_{\text{ABS}}$ outputs $\perp$. Observe that the programs $\mathcal{V}_{\text{ABS}}$ and $\mathcal{V}'_{\text{ABS}}$ are clearly functionally identical since the punctured PPRF key behaves identically to the full PPRF key on all inputs $x \neq x^*$. For the same reason, for all the signing keys given to $\mathcal{A}$, the programs $\mathcal{P}_{\text{ABS}}$ and $\mathcal{P}'_{\text{ABS}}$ are also functionally identical since $\mathcal{A}$ is allowed to request signing keys for only those signing policies that does not accept x^*. Thus, by the security of IO, which stipulates that obfuscations of functionally identical programs are computationally indistinguishable, the modified experiment is computationally indistinguishable from the original one. After that, we apply the pseudorandomness at punctured point property of PPRF to change the pseudorandom string $\hat{r}_{\text{SIG}}$ hardwired within $\mathcal{V}'_{\text{ABS}}$ to a uniformly random one. This modification essentially ensures that a perfectly distributed SIG signing key-verification key pair gets associated to x^*. Note that once this alteration is made, we can directly prove the unforgeability of our ABS scheme relying on the unforgeability property of SIG.

Problem with Accommodating Unbounded-Length Signing Attributes

Now, observe that we want to consider signing policies as TMs and signing attribute strings of arbitrary polynomial length. Then we must represent the signing and verifying programs in the above ABS construction as TMs, and use some IO scheme for TMs to obfuscate those programs. Several recent works have built IO candidates for TMs [21, 3]. However, we cannot directly use such an IO for TM scheme in our ABS construction sketched above. Observe that IO guarantees indistinguishability of two functionally equivalent programs if and only if they have the same size. In our proof of unforgeability, we use IO to switch between two sets of programs, where one set of programs have the full PPRF key κ hardwired in

it, while the other have the punctured PPRF key $\kappa\{x^*\}$ hardwired. Thus, in order to use IO security, we must ensure that programs having κ hardwired, i.e., those used in the actual construction, have the same size as those having $\kappa\{x^*\}$. Unfortunately, in the existing PPRF construction [6], namely, the tree-based PPRF construction derived from the PRF construction of [14], the size of punctured PPRF key depends linearly on the length of the punctured point. This means that if we are to upper bound the size of the programs to be used in the real construction, we must put an upper bound on the length of the signing attributes, which we want to avoid.

In order to overcome the length bounding issue discussed above, a natural direction is to use some compressing tool, e.g., a collision-resistant hash function $\mathcal{H}$ to map arbitrary-length signing attribute strings to a fixed length. Concretely, let us define the SIG signing key-verification key pair associated with a signing attribute string x to be the one obtained by running the setup algorithm of SIG using the pseudorandom string obtained as the output of the PPRF not on x itself, but on $\mathcal{H}(x)$, and accordingly modify our signing programs $\mathcal{P}_{\text{ABS}}$ and verifying program $\mathcal{V}_{\text{ABS}}$ to evaluate the PPRF on the hash values of the input signing attribute strings rather than applying on signing attribute strings themselves directly. While this alteration would allow us to support signing attribute strings of arbitrary polynomial length, it would pose new challenges. Observe that in order to use a strategy similar to our earlier proof of unforgeability, we must modify the real signing and verifying programs having the full PPRF key κ into ones having the punctured PPRF key $\kappa\{h^*\}$ punctured at $h^* = \mathcal{H}(x^*)$. However, unlike the previous scenario, now we cannot rely on IO security to perform this transformation since the real and modified sets of programs are not functionally identical in general. For instance, consider the real signing program $\mathcal{P}_{\text{ABS}}$ (with the full PPRF key κ hardwired) and the modified program $\mathcal{P}'_{\text{ABS}}$ (with the punctured PPRF key $\kappa\{h^*\}$ hardwired) included within a signing key provided to the adversary $\mathcal{A}$ in the real and modified unforgeability experiment respectively. Now, consider some signing attribute string $x \neq x^*$ which is accepted by the associated signing policy, and for which $\mathcal{H}(x) = h^*$. Then, on input x, $\mathcal{P}_{\text{ABS}}$ would output the SIG signing key-verification key pair associated with x (which are the same as those associated with x^* by definition) since it has the full PPRF key, while $\mathcal{P}'_{\text{ABS}}$ would output $\perp$ since it has the punctured PPRF key.

To settle the above issue, one possible option could be to resort to stronger forms of obfuscation, e.g., differing-input obfuscation (DIO) [4, 7] or its weaker variant, namely, public-coin differing-input obfuscation (PCDIO) [16]. However, the existence of DIO or its variants are highly strong knowledge-type assumptions, and basing security of cryptographic construction on such assumptions are largely considered to be risky. In fact, DIO was already shown to be implausible to exist [11]. On the contrary, there is no known impossibility or implausibility result against IO. On the contrary, as we have already mentioned, a steady progress has taken place in the last few years towards building IO candidates. Therefore, we search for other techniques so as to base the security of our ABS scheme on IO.

Approach along the Lines of [21, 10] and its Limitation

At this point, we look into the recent work of Deshpande et al. [10], where they have employed the elegant techniques of Koppula et al. [21] to design a constrained pseudorandom function (CPRF) for constraints representable as TMs of arbitrary description size that can handle inputs of arbitrary polynomial length, based on IO for circuits and PPRF. The notion of CPRFs [6] is a generalization of the notion of PPRFs. More precisely, a CPRF is an augmentation of a standard PRF with an additional constrain algorithm which enables a party holding a PRF key to derive constrained keys corresponding to specific constraint predicates. Such a constrained key allows the evaluation of the PRF at all points of the input domain that are accepted by the predicate. However, given a set of constrained keys, the PRF evaluations still remain indistinguishable from random on all the inputs not covered by those constraint predicates.

The high level idea behind the CPRF construction of Deshpande et al. [10] is as follows: Similar to our approach, to produce the CPRF output, their construction uses a PPRF and a compressing tool that maps arbitrarily long inputs to fixed size ones. However, the compressing tool that they have used is not a mere hash function, but a more advanced one, namely a (prefixed) positional accumulator [21], which possesses several additional IO-friendly properties that a usual hash function does not. Roughly speaking, a positional accumulator is a cryptographic data structure that maintains two values, namely, a storage value and an accumulator value. The storage value is allowed to grow comparatively large, while the accumulator value is constrained to be short. Message symbols can be written to various positions in the underlying storage, and new accumulated values can be computed dynamically, knowing only the previous accumulator value and the newly written symbol together with its position in the data structure. Moreover, there are additional helper algorithms which essentially allow a party who is maintaining the full

storage to help a more restricted party maintaining only the accumulator value recover the data currently written at an arbitrary location. However, the helper is not necessarily trusted. So, it should additionally provide a short proof to convince the accumulator-value-maintaining party about the correctness of the symbols being read. In the CPRF construction of Deshpande et al. [10], a master CPRF key consists of a key κ for the underlying PPRF and a set of public parameters $\mathsf{PP}_{\mathsf{ACC}}$ of the positional accumulator. The CPRF evaluation on some input x is simply the output of the PPRF with key κ on input w_{INP}, where w_{INP} is the accumulation of the bits of x using $\mathsf{PP}_{\mathsf{ACC}}$.

A constrained key of the CPRF corresponding to some TM M, comprises of $\mathsf{PP}_{\mathsf{ACC}}$ along with two programs $\mathcal{P}_1$ and $\mathcal{P}_{\mathsf{CPRF}}$, which are obfuscated using IO. The first program $\mathcal{P}_1$, also known as the initial signing program, takes as input an accumulator value and outputs a signature on it together with the initial state and header position of the TM M. The second program $\mathcal{P}_{\mathsf{CPRF}}$, also called the next step program, has the PPRF key κ hardwired in it. It takes as input a state and header position of M, along with an input symbol and an accumulator value. It essentially computes the next step function of M on the input state-symbol pair, and eventually outputs the proper CPRF value, if M reaches the accepting state. The program $\mathcal{P}_{\mathsf{CPRF}}$ also performs certain authenticity checks before computing the next step function of M in order to prevent illegal inputs. For this purpose, $\mathcal{P}_{\mathsf{CPRF}}$ additionally takes as input a signature on the input state, header position, and accumulator value, together with a proof for the positional accumulator. The program $\mathcal{P}_{\mathsf{CPRF}}$ verifies the signature in order to ensure authenticity, as well as checks the accumulator proof to get convinced that the input symbol is indeed the one placed at the input header position of the underlying storage of the input accumulator value. If all these verifications pass, then $\mathcal{P}_{\mathsf{CPRF}}$ determines the next state and header position of M, as well as the new symbol that needs to be written to the input header position. The program $\mathcal{P}_{\mathsf{CPRF}}$ then updates the accumulator value by placing the new symbol at the input header position, as well as signs the updated accumulator value along with the computed next state and header position of M. In order to deal with the security proof, the signature scheme used by the two programs is a special type of IO-compatible signature, namely, splittable signature.

Evaluating the CPRF on some input x using a constrained key corresponding to some TM M, consists of two steps. In the first step, the evaluator computes the accumulation w_{INP} of the bits of x using $\mathsf{PP}_{\mathsf{ACC}}$, which are also included in the constrained key, and then obtains a signature on w_{INP} together with the initial state and header position of M, by running the program $\mathcal{P}_1$. The second step is to repeatedly run the program $\mathcal{P}_{\mathsf{CPRF}}$, each time on input the current accumulator value, current state and header position of M, along with the signature on them. Additionally, in each iteration, the evaluator also feeds w_{INP} to $\mathcal{P}_{\mathsf{CPRF}}$. The iteration is continued until the program $\mathcal{P}_{\mathsf{CPRF}}$ either outputs the proper CPRF value, namely, the evaluation of the PPRF with key κ on w_{INP}, or the designated symbol $\perp$ indicating failure.

We attempt to follow the above approach in designing our ABS scheme. More precisely, we augment the above CPRF construction as follows: We treat the signing attribute strings like the inputs of the above CPRF construction. Just as their CPRF master key, our master signing key consists of a PPRF key κ and a set of public parameters $\mathsf{PP}_{\mathsf{ACC}}$ of the positional accumulator. We define the SIG signing key-verification key pair associated with a signing attribute string x as those obtained by running the setup algorithm of SIG using the randomness obtained by evaluating the underlying PPRF with key κ on w_{INP}, where w_{INP} is the accumulation of the bits of x using $\mathsf{PP}_{\mathsf{ACC}}$. Similar to their constrained keys, our signing key corresponding to some TM M would comprise of two IO-obfuscated program $\mathcal{P}_1$ and $\mathcal{P}_{\mathsf{ABS}}$. The functionality of the program $\mathcal{P}_1$ would be exactly same as the program $\mathcal{P}_1$ in the above CPRF construction, whereas the functionality of $\mathcal{P}_{\mathsf{ABS}}$ would be an augmentation of that of $\mathcal{P}_{\mathsf{CPRF}}$ in the above CPRF construction. Precisely, in case reaching to the accepting state, $\mathcal{P}_{\mathsf{ABS}}$ first computes the PPRF with key κ on input w_{INP} just like $\mathcal{P}_{\mathsf{CPRF}}$. After that it generates and outputs a SIG signing key-verification key pair by running the setup algorithm of SIG using the computed pseudorandom string. Our ABS public parameters would contain $\mathsf{PP}_{\mathsf{ACC}}$ along with the IO-obfuscated verifying program $\mathcal{V}_{\mathsf{ABS}}$ that has the same functionality as earlier, except that instead of taking as input a signing attribute string directly, it now take as input an accumulator value and applies the PPRF on the input accumulator value.

In order to sign a message under some signing attribute string accepted by the TM embedded in its signing key, a signer first follows similar steps as those involved in the CPRF evaluation procedure described above to obtain the SIG signing key-verification key pair associated with the signing attribute string. After that, the signer signs the message using the obtained SIG signing key. The signature verification process remains exactly same as earlier, except that instead of directly inputting the claimed signing attribute string to $\mathcal{V}_{\mathsf{ABS}}$, now a verifier have to accumulate the bits of the claimed signing attribute string using $\mathsf{PP}_{\mathsf{ACC}}$, and input the accumulated value to $\mathcal{V}_{\mathsf{ABS}}$.

While the above strategy appears to be sound, there still remain some subtle issues. Observe that To handle the positional accumulator related verifications and updations, just like the programs $\mathcal{P}_{\text{CPRF}}$ in the CPRF construction of Deshpande et al. [10], the programs $\mathcal{P}_{\text{ABS}}$ must have PP_{ACC} hardwired in them. During the course of the unforgeability proof, we have to modify the signing keys given to the adversary $\mathcal{A}$ to embed the punctured PPRF key $\text{K}\{w^*_{\text{INP}}\}$ punctured at w^*_{INP} instead of the full PPRF key K. Here, w^*_{INP} is the accumulation of the bits of the challenge signing attribute string x^*, committed by $\mathcal{A}$ at the beginning of the experiment, using PP_{ACC} included within the public parameters given to the adversary $\mathcal{A}$. In order to make this substitution, it is to be ensured that the programs $\mathcal{P}_{\text{ABS}}$ included in those signing keys always outputs $\perp$ for signing attribute strings corresponding to w^*_{INP} even if reaching the accepting state. As usual, we would carry out the transformation one signing key at a time through multiple hybrid steps. Now, suppose for transforming the signing keys we attempt to follow a strategy similar to that of [21] or [10]. Let the total number of signing keys queried by $\mathcal{A}$ be $\hat{q}_{\text{KEY}}$. Consider the transformation of the ν^{th} signing key ($1 \leq \nu \leq \hat{q}_{\text{KEY}}$) corresponding to the TM $M^{(\nu)}$ that runs on the challenge signing attribute string x^* for $t^{*(\nu)}$ steps and reaches the rejecting state. In the course of transformation, the program $\mathcal{P}^{(\nu)}_{\text{ABS}}$ contained in the ν^{th} signing key would first be altered to one that always outputs $\perp$ for inputs corresponding to w^*_{INP} within the first $t^{*(\nu)}$ steps. Towards accomplishing this transition, in successive hybrids, the steps of execution of $M^{(\nu)}$ on x^* would be repeatedly programmed and unprogrammed within $\mathcal{P}^{(\nu)}_{\text{ABS}}$ taking the help of IO. In order to perform this operation using IO, at various stages, we need to guarantee program functional equivalence, and for that we need to generate PP_{ACC} in read/write enforcing mode, certain special statistically binding modes indistinguishable from the normal setup mode. However, in the prefixed version of positional accumulator employed in [10] or in [21], to setup PP_{ACC} in read/write-enforcing mode, we require the entire sequence of symbol-position pairs arising from iteratively running $M^{(\nu)}$ on x^* up to the step we are programming in. This was not a problem for [21] or [10] since in their security model the adversary $\mathcal{A}$ was bounded to declare the TM queries prior to setting up the system. On the contrary, in our unforgeability experiment, $\mathcal{A}$ is allowed to adaptively submit signing key queries corresponding to signing policy TMs of its choice throughout the experiment. In such a case, we would be able to determine those symbol-position pairs *only after receiving* the ν^{th} queried TM $M^{(\nu)}$ from $\mathcal{A}$. However, we would require PP_{ACC} while creating the signing keys queried by $\mathcal{A}$ before making the ν^{th} signing key query, and even for preparing the public parameters. Thus, it is immediate that we must generate PP_{ACC} *prior to receiving* the ν^{th} signing key query from $\mathcal{A}$. This is clearly *impossible* as setting PP_{ACC} in read/write enforcing mode requires the knowledge of the TM $M^{(\nu)}$, which is *not available* before the ν^{th} signing key query of $\mathcal{A}$.

Approach to Solve the Limitation of [21,10] using Adaptive Positional Accumulator and the Associated Challenge

Observe that a set of public parameters of the positional accumulator must be included within each signing key. This is mandatory due to the required updatability feature of positional accumulator, which is indispensable to keep track of the current situation while running the program $\mathcal{P}_{\text{ABS}}$ iteratively in the course of signing a message under some signing attribute string. The root cause of the problem discussed above, however, is the use of a single set of public parameters PP_{ACC} of the positional accumulator throughout the system. Observe that this problem would immediately disappear if we use a more advanced variant of positional accumulator, namely, adaptive positional accumulator [8] as opposed to the prefixed ones employed in [21, 10]. Adaptive positional accumulators do not require the history of computation for setting up the public parameters. More precisely, in case of adaptive positional accumulators, there are two separate algorithms, one for generating the public parameters needed for updating the storage and accumulator values, and the other for creating the verification key for verifying the correctness of accumulator proofs. In order to get the enforcing properties, it is only sufficient to generate the verification key in the enforcing mode. Thus, we can generate a single system-wide set of public parameters of positional accumulator, while hardwire a different verification key of positional accumulator within the program $\mathcal{P}_{\text{ABS}}$ contained in each signing key. Then, in our proof of unforgeability, when modifying the signing keys, we can set the accumulator verification key associated with the signing key being modified in the enforcing mode without affecting other signing keys or the public parameter. However, in case of adaptive positional accumulators, unlike the prefixed variant, there is no helper algorithm by which a storage-maintaining party could assist a party, who is only maintaining the accumulator value, to update the accumulator value following an update in the storage. This means that we should also input the entire accumulator storage, which is of a potentially exponential size, to $\mathcal{P}_{\text{ABS}}$ in each iteration as otherwise the program cannot update the accumulator value after executing the next step function of the embedded

TM. This would again pose new challenges. Note however that Canetti et al. [8] could manage to update the accumulator value outside the obfuscated programs roughly since they dealt with RAM machines, and unlike TMs, RAM machines read from the same memory location in the next iteration where they have written to in the previous iteration. More precisely, a TM reads the symbol at the current header position, writes a possibly new symbol at that location, and moves its header to a new location for the next iteration. On the other hand, a RAM machine reads a symbol from the current header position, and writes a possibly new symbol at some possibly different location by moving its header to that location, where from it starts the next iteration by reading the value just written down.

Our Solution

Instead of going into that, we try to explore whether it is possible to resolve the problem arising out of the approach of [21, 10] without altering any cryptographic tool used in [21, 10]. We have already noticed that the problem in the approach of [21, 10] is the use of a single set of public parameters of positional accumulator throughout the system. Therefore, we attempt to assign a fresh set of public parameters of the positional accumulator to each signing key. However, for compressing the signing attribute strings to a fixed length, on which the PPRF can be applied to produce the pseudorandom strings specifying the SIG signing key-verification key pairs associated with those signing attribute strings, we need a system-wide compressing tool. We employ a somewhere statistically-binding (SSB) hash function for this purpose. However, note that this does not mean the introduction of any new tool or assumption since one can always replace an SSB hash function with a prefixed positional accumulator. In fact, SSB hash functions, introduced in [15], has a similar type of property as positional accumulator, although is less functional than it. Roughly speaking, just like an ordinary hash function, an SSB hash can be used to create a short digest of some long string. However, in case of SSB hashes, a hashing key is created by specifying a special binding index and the generated hashing key gets the property that the hash value of some string created with the hashing key is statistically binding for the specified index, meaning that the hash value completely determines the symbol of the hashed input at that index. Moreover, it is possible to prove that the input string underlying a given hash value contains a specific symbol at a particular index, by providing a short opening value.

Our idea is that while signing a message under some signing attribute string x using its legitimate signing key for some TM M, the signer first computes the hash value h by hashing x using the system-wide SSB hash key, which is part of the ABS public parameters. The signer also computes the accumulator value w_{INP} by accumulating the bits of x using the public parameters of positional accumulator, specific to its signing key. Then, using the obfuscated initial signing program $\mathcal{P}_1$ included in its signing key, the signer will obtain a signature on w_{INP} along with the initial state and header position of M. Finally, the signer will repeatedly run the obfuscated next step program $\mathcal{P}_{\text{ABS}}$ included in its signing key, each time giving as input all the quantities as earlier, except that it would now have to feed the SSB hash value h in place of w_{INP} in each iteration. This is because, in case $\mathcal{P}_{\text{ABS}}$ reaches the accepting state, it would require h to apply the PPRF for producing the SIG signing key-verification key pair associated with x. The same change would also apply to the public verifying program $\mathcal{V}_{\text{ABS}}$, namely, it would also take as input the SSB hash value of a signing attribute string in place of the accumulator value obtained by accumulating its bits.

However, this approach is not completely sound yet. Observe that, a possibly malicious signer can compute the SSB hash value h on the signing attribute string x, with respect to which it wishes to generate a signature despite of the fact that its signing policy TM M does not accepts it, but initiates the computation by accumulating the bits of only a substring of x or some entirely different signing attribute string, which is accepted by M. To prevent such malicious behavior, we include another IO-obfuscated program $\mathcal{P}_2$ within the signing key, we call the accumulating program, whose purpose is to restrict the signer from accumulating the bits of a different signing attribute string rather than the hashed one. The program $\mathcal{P}_2$ takes as input an SSB hash value h, an index i, a symbol, an accumulator value, a signature on the input accumulator value (along with the initial state and header position of M), and an opening value for SSB. The program $\mathcal{P}_2$ verifies the signature, and also checks whether the input symbol is indeed present at the index i of the string that has been hashed to form h, using the input opening value. If all of these verifications pass, then $\mathcal{P}_2$ updates the input accumulator value by writing the input symbol at the i^{th} position of the accumulator storage, and signs the updated accumulator value (along with the initial state and header position of M). The signature used by $\mathcal{P}_2$ is also a splittable signature that facilitates the security proof. The obfuscated initial signing program $\mathcal{P}_1$ included in the signing key is also modified

to take as input a hash value, and output a signature on the accumulator value corresponding to the empty accumulator storage together with the initial state and header position of M.

Moreover, for forbidding the signer from performing the computation by accumulating an M-accepted substring of the hashed input, we define the SIG signing key-verification key pair associated with a signing attribute string as the output of the setup algorithm of SIG using the pseudorandom string generated by applying the PPRF on the pair (hash value, length) of the signing attribute string in stead of just the hash value. Note that, without loss of generality, we can set the upper bound of the length of signing attribute strings to be 2^λ, where λ is the underlying security parameter, in view of the fact that by suitably choosing λ we can accommodate signing attribute strings of any polynomial length. Since the lengths of the attribute strings are bounded by 2^λ, the lengths can be expressed as bit strings of size λ. Hence, the total size of the hash value-length pair corresponding to a signing attribute string would be bounded, and the problem of arbitrarily large punctured PPRF key size would not appear. However, now the obfuscated next step programs $\mathcal{P}_{\text{ABS}}$ included in our signing keys, must also take as input the length of the signing attribute strings for applying the PPRF if reaching to the accepting state.

Thus, the signing procedure of our ABS scheme becomes the following: to sign a message under some signing attribute string using its legitimate signing key corresponding to some TM M, a signer first hash the signing attribute string with the system-wide SSB hash key. The signer also obtains a signature on the empty accumulator value, by running the obfuscated initial signing program $\mathcal{P}_1$ on input the computed hash value. Next, it repeatedly runs the obfuscated accumulating program $\mathcal{P}_2$ to authentically accumulate the bits of the hashed signing attribute string. Finally, it runs the obfuscated next step program $\mathcal{P}_{\text{ABS}}$ iteratively on the current accumulator value along with other legitimate inputs, until it obtains either the SIG signing key-verification key pair associated with the signing attribute string under consideration or $\perp$. Once it obtains the SIG signing key-verification key pair associated with the signing attribute string, it simply signs the message using the SIG signing key, and outputs the SIG verification key-signature pair as the ABS signature on the message.

Notice that the problem with enforcing the public parameters of the positional accumulator while transforming the adaptively queried signing keys will not appear in our case as we have assigned a separate set of public parameters of positional accumulator to each signing key. However, our actual proof of unforgeability involves many subtleties that are difficult to describe with this high level description, and is provided in full details in the sequel. We would only like to mention here that to cope up with certain issues in the proof, another IO-obfuscated program $\mathcal{P}_3$ is also included within the signing keys, we call the signature changing program, that changes the splittable signature obtained from $\mathcal{P}_2$ on the accumulation of the bits of the signing attribute string, before starting the iterative computation with the obfuscated next step program $\mathcal{P}_{\text{ABS}}$.

We follow the same novel technique introduced in [10] for handling the tail hybrids in the final stage of transformation of the signing keys in our unforgeability experiment. Note that as in [10], we consider TMs which run for at most $T = 2^\lambda$ steps on any input. Unlike [21], Deshpande et al. [10] have devised an elegant approach to obtain an end to end polynomial reduction to the security of IO for the tail hybrids by means of an injective pseudorandom generator (PRG). We directly adopt that technique to deal with the tail hybrids in our unforgeability proof. Please refer to [10] for a high level overview of the approach.

2 Preliminaries

Here we give the necessary background on various cryptographic tools which we will be using in the sequel. Let $\lambda \in \mathbb{N}$ denotes the security parameter. For $n \in \mathbb{N}$ and $a, b \in \mathbb{N} \cup \{0\}$ (with $a < b$), we let $[n] = \{1, \ldots, n\}$ and $[a, b] = \{a, \ldots, b\}$. For any set S, $v \xleftarrow{\$} S$ represents the uniform random variable on S. For a randomized algorithm $\mathcal{R}$, we denote by $\psi = \mathcal{R}(v; \rho)$ the random variable defined by the output of $\mathcal{R}$ on input v and randomness ρ, while $\psi \xleftarrow{\$} \mathcal{R}(v)$ has the same meaning with the randomness suppressed. Also, if $\mathcal{R}$ is a deterministic algorithm $\psi = \mathcal{R}(v)$ denotes the output of $\mathcal{R}$ on input v. We will use the alternative notation $\mathcal{R}(v) \to \psi$ as well to represent the output of the algorithm $\mathcal{R}$, whether randomized or deterministic, on input v. For any string $s \in \{0,1\}^*$, $|s|$ represents the length of the string s. For any two strings $s, s' \in \{0,1\}^*$, $s \| s'$ represents the concatenation of s and s'. A function negl is *negligible* if for every integer c, there exists an integer k such that for all $\lambda > k$, $|\mathsf{negl}(\lambda)| < 1/\lambda^c$.

2.1 Turing Machines

A Turing machine (TM) M is a 7-tuple $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$ with the following semantics:

- Q: The finite set of possible states of M.
- Σ_{INP}: The finite set of input symbols.
- Σ_{TAPE}: The finite set of tape symbols such that $\Sigma_{\text{INP}} \subset \Sigma_{\text{TAPE}}$ and there exists a special blank symbol '$\_$' $\in \Sigma_{\text{TAPE}} \backslash \Sigma_{\text{INP}}$.
- $\delta : Q \times \Sigma_{\text{TAPE}} \to Q \times \Sigma_{\text{TAPE}} \times \{+1, -1\}$: The transition function of M.
- $q_0 \in Q$: The designated start state.
- $q_{\text{AC}} \in Q$: The designated accept state.
- $q_{\text{REJ}} (\neq q_{\text{AC}}) \in Q$: The distinguished reject state.

For any $t \in [T = 2^\lambda]$, we define the following variables for M, while running on some input (without the explicit mention of the input in the notations):

- $\text{POS}_{M,t}$: An integer which denotes the position of the header of M after the t^{th} step. Initially, $\text{POS}_{M,0} = 0$.
- $\text{SYM}_{M,t} \in \Sigma_{\text{TAPE}}$: The symbol stored on the tape at the $\text{POS}_{M,t}{}^{\text{th}}$ location.
- $\text{SYM}_{M,t}^{(\text{WRITE})} \in \Sigma_{\text{TAPE}}$: The symbol to be written at the $\text{POS}_{M,t-1}{}^{\text{th}}$ location during the t^{th} step.
- $\text{ST}_{M,t} \in Q$: The state of M after the t^{th} step. Initially, $\text{ST}_{M,0} = q_0$.

At each time step, theTM M reads the tape at the header position and based on the current state, computes what needs to be written on the tape at the current header location, the next state, and whether the header must move left or right. More formally, let $(q, \zeta, \beta \in \{+1, -1\}) = \delta(\text{ST}_{M,t-1}, \text{SYM}_{M,t-1})$. Then, $\text{ST}_{M,t} = q, \text{SYM}_{M,t}^{(\text{WRITE})} = \zeta$, and $\text{POS}_{M,t} = \text{POS}_{M,t-1} + \beta$. M accepts at time t if $\text{ST}_{M,t} = q_{\text{AC}}$. In this paper we consider $\Sigma_{\text{INP}} = \{0, 1\}$ and $\Sigma_{\text{TAPE}} = \{0, 1, \_\}$. Given any TM M and string $x \in \{0,1\}^*$, we define $M(x) = 1$, if M accepts x within T steps, and 0, otherwise.

2.2 Indistinguishability Obfuscation

Definition 2.1 (Indistinguishability Obfuscation: IO [12]). An indistinguishability obfuscator (IO) $\mathcal{IO}$ for a circuit class $\{\mathbb{C}_\lambda\}_\lambda$ is a probabilistic polynomial-time (PPT) uniform algorithm satisfying the following conditions:

► **Correctness**: $\mathcal{IO}(1^\lambda, C)$ preserves the functionality of the input circuit C, i.e., for any $C \in \mathbb{C}_\lambda$, if we compute $C' = \mathcal{IO}(1^\lambda, C)$, then $C'(\upsilon) = C(\upsilon)$ for all inputs υ.

► **Indistinguishability**: For any security parameter λ and any two circuits $C_0, C_1 \in \mathbb{C}_\lambda$ with same functionality, the circuits $\mathcal{IO}(1^\lambda, C_0)$ and $\mathcal{IO}(1^\lambda, C_1)$ are computationally indistinguishable. More precisely, for all (not necessarily uniform) PPT adversaries $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$, there exists a negligible function negl such that, if

$$\mathsf{Pr}\big[(C_0, C_1, \xi) \xleftarrow{\$} \mathcal{D}_1(1^\lambda) \ : \ \forall\, \upsilon, C_0(\upsilon) = C_1(\upsilon)\big] \geq 1 - \mathsf{negl}(\lambda),$$
$$\text{then } \big|\mathsf{Pr}\big[\mathcal{D}_2(\xi, \mathcal{IO}(1^\lambda, C_0)) = 1\big] - \mathsf{Pr}\big[\mathcal{D}_2(\xi, \mathcal{IO}(1^\lambda, C_1)) = 1\big]\big| \leq \mathsf{negl}(\lambda).$$

We remark that the two distinct algorithms $\mathcal{D}_1$ and $\mathcal{D}_2$, which pass state ξ, can be viewed equivalently as a single stateful algorithm $\mathcal{D}$. In this paper we employ the latter approach, although here we present the definition as it appears in [12]. When clear from the context, we will drop 1^λ as an input to $\mathcal{IO}$.

The circuit class we are interested in are polynomial-size circuits, i.e., when $\mathbb{C}_\lambda$ is the collection of all circuits of size at most λ. This circuit class is denoted by $\mathsf{P/poly}$. The first candidate construction of IO for $\mathsf{P/poly}$ was presented by Garg et al. [12] in 2013. Their construction uses nonstandard instance dependent assumption on graded multilinear encodings. Since then, there has been a rapid progress towards designing IO from better understood cryptographic tools and complexity assumptions. Very recently, a series of exciting works [2, 22, 1, 17, 19, 13, 18] have finally provided an IO candidate based on the sub-exponential security of four well-studied computational assumptions, namely, learning with errors (LWE), learning parity with noise (LPN), existence of boolean pseudorandom generators (PRG) in NC^0, and symmetric external Diffie-Hellman (SXDH).

2.3 IO-Compatible Cryptographic Primitives

In this section, we describe the notions of certain IO-friendly cryptographic tools used in our ABS construction.

2.3.1 Puncturable Pseudorandom Function

A puncturable pseudorandom function, introduced by Sahai and Waters [27], is a pseudorandom function (PRF) with the additional property that given a master PRF key, it is possible to derive punctured PRF keys that enable the evaluation of the PRF at all points of the input domain except the punctured points, whereas given such a punctured key, the PRF output still remains pseudorandom at the punctured point.

Definition 2.2 (Puncturable Pseudorandom Function: PPRF [27]). A puncturable pseudorandom function (PPRF) $\mathcal{F} : \mathcal{K}_{\text{PPRF}} \times \mathcal{X}_{\text{PPRF}} \to \mathcal{Y}_{\text{PPRF}}$ consists of an additional punctured key space $\mathcal{K}_{\text{PPRF-PUNC}}$ other than the usual key space $\mathcal{K}_{\text{PPRF}}$ and the PPT algorithms ($\mathcal{F}$.Setup, $\mathcal{F}$.Eval, $\mathcal{F}$.Puncture, $\mathcal{F}$.Eval-Punctured) described below. Here, $\mathcal{X}_{\text{PPRF}} = \{0,1\}^{\ell_{\text{PPRF-INP}}}$ and $\mathcal{Y}_{\text{PPRF}} = \{0,1\}^{\ell_{\text{PPRF-OUT}}}$, where $\ell_{\text{PPRF-INP}}$ and $\ell_{\text{PPRF-OUT}}$ are polynomials in the security parameter λ,

- $\mathcal{F}.\mathsf{Setup}(1^\lambda) \to K$: The setup authority takes as input the security parameter 1^λ and uniformly samples a PPRF key $K \in \mathcal{K}_{\text{PPRF}}$.
- $\mathcal{F}.\mathsf{Eval}(K, x) \to r$: The setup authority takes as input a PPRF key $K \in \mathcal{K}_{\text{PPRF}}$ along with an input $x \in \mathcal{X}_{\text{PPRF}}$. It outputs the PPRF value $r \in \mathcal{Y}_{\text{PPRF}}$ on x. For simplicity, we will represent by $\mathcal{F}(K, x)$ the output of this algorithm.
- $\mathcal{F}.\mathsf{Puncture}(K, x) \to K\{x\}$: Taking as input a PPRF key $K \in \mathcal{K}_{\text{PPRF}}$ along with an element $x \in \mathcal{X}_{\text{PPRF}}$, the setup authority outputs a punctured key $K\{x\} \in \mathcal{K}_{\text{PPRF-PUNC}}$.
- $\mathcal{F}.\mathsf{Eval\text{-}Puncured}(K\{x\}, x') \to r$ or $\perp$: An evaluator takes as input a punctured key $K\{x\} \in \mathcal{K}_{\text{PPRF-PUNC}}$ along with an input $x' \in \mathcal{X}_{\text{PPRF}}$. It outputs either a value $r \in \mathcal{Y}_{\text{PPRF}}$ or a distinguished symbol $\perp$ indicating failure. For simplicity, we will represent by $\mathcal{F}(K\{x\}, x')$ the output of this algorithm.

The algorithms $\mathcal{F}$.Setup and $\mathcal{F}$.Puncture are randomized, whereas, $\mathcal{F}$.Eval and $\mathcal{F}$.Eval-Punctured are deterministic.

► **Correctness under Puncturing**: Consider any security parameter λ, $K \in \mathcal{K}_{\text{PPRF}}$, $x \in \mathcal{X}_{\text{PPRF}}$, and $K\{x\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, x)$. Then it must hold that

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

► **Selective Pseudorandomness at Punctured Points**: This property of a PPRF is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ submits a challenge input $x^* \in \mathcal{X}_{\text{PPRF}}$ to $\mathcal{C}$.
- $\mathcal{C}$ chooses uniformly at random a PPRF key $K^* \xleftarrow{\$} \mathcal{K}_{\text{PPRF}}$ and a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. It computes the punctured key $K^*\{x^*\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^*, x^*)$. If $\hat{b} = 0$, it sets $r^* = \mathcal{F}(K^*, x^*)$. Otherwise, it selects $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. It sends back $(K^*\{x^*\}, r^*)$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The PPRF $\mathcal{F}$ is selectively pseudorandom at punctured points if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\mathcal{F},\text{SEL-PR}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Boneh and Waters [6], have shown that the tree-based PRF constructed by Goldreich et al. [14] can be readily modified to build a PPRF from one-way functions.

2.3.2 Somewhere Statistically Binding Hash Function

We provide the definition of somewhere statistically binding hash function as defined by Hubacek et al. [15]. A somewhere statistically binding hash can be used to create a short digest of some long string. A hashing key is created by specifying a special binding index and the generated hashing key gets the property that the hash value of some string created with the hashing key is statistically binding for the specified index, meaning that the hash value completely determines the symbol of the hashed input at that index. In other words, even if some hash value has several pre-images, all of those pre-images

agree in the symbol at the specified index. The index on which the hash is statistically binding should remain computationally hidden given the hashing key. Moreover, it is possible to prove that the input string underlying a given hash value contains a specific symbol at a particular index, by providing a short opening value.

Definition 2.3 (Somewhere Statistically Binding Hash Function: SSB Hash [15,25]). A somewhere statistically binding (SSB) hash consists of the PPT algorithms ($\mathsf{SSB.Gen}, \mathcal{H}, \mathsf{SSB.Open}, \mathsf{SSB.Verify}$) along with a block alphabet $\Sigma_{\text{SSB-BLK}} = \{0,1\}^{\ell_{\text{SSB-BLK}}}$, output size $\ell_{\text{SSB-HASH}}$, and opening space $\Pi_{\text{SSB}} = \{0,1\}^{\ell_{\text{SSB-OPEN}}}$, where $\ell_{\text{SSB-BLK}}, \ell_{\text{SSB-HASH}}, \ell_{\text{SSB-OPEN}}$ are some polynomials in the security parameter λ. The algorithms have the following syntax:

- $\mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}}, i^*) \rightarrow \text{HK}$: The setup authority takes as input the security parameter 1^λ, an integer $n_{\text{SSB-BLK}} \leq 2^\lambda$ representing the maximum number of blocks that can be hashed, and an index $i^* \in [0, n_{\text{SSB-BLK}} - 1]$ and publishes a public hashing key HK.
- $\mathcal{H}_{\text{HK}} : x \in \Sigma_{\text{SSB-BLK}}^{n_{\text{SSB-BLK}}} \rightarrow h \in \{0,1\}^{\ell_{\text{SSB-HASH}}}$: This is a deterministic function that has the hash key HK hardwired. A user runs this function on input $x = x_0 \| \ldots \| x_{n_{\text{SSB-BLK}}-1} \in \Sigma_{\text{SSB-BLK}}^{n_{\text{SSB-BLK}}}$ to obtain as output $h = \mathcal{H}_{\text{HK}}(x) \in \{0,1\}^{\ell_{\text{SSB-HASH}}}$.
- $\mathsf{SSB.Open}(\text{HK}, x, i) \rightarrow \pi_{\text{SSB}}$: Taking as input the hash key HK, input $x \in \Sigma_{\text{SSB-BLK}}^{n_{\text{SSB-BLK}}}$, and an index $i \in [0, n_{\text{SSB-BLK}} - 1]$, a user creates an opening $\pi_{\text{SSB}} \in \Pi_{\text{SSB}}$.
- $\mathsf{SSB.Verify}(\text{HK}, h, i, u, \pi_{\text{SSB}}) \rightarrow \hat{\beta} \in \{0,1\}$: On input a hash key HK, a hash value $h \in \{0,1\}^{\ell_{\text{SSB-HASH}}}$, an index $i \in [0, n_{\text{SSB-BLK}} - 1]$, a value $u \in \Sigma_{\text{SSB-BLK}}$, and an opening $\pi_{\text{SSB}} \in \Pi_{\text{SSB}}$, a verifier outputs a bit $\hat{\beta} \in \{0,1\}$.

The algorithms SSB.Gen and SSB.Open are randomized, while the algorithm SSB.Verify is deterministic.

► **Correctness**: For any security parameter λ, integer $n_{\text{SSB-BLK}} \leq 2^\lambda$, $i, i^* \in [0, n_{\text{SSB-BLK}} - 1]$, $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}}, i^*)$, $x \in \Sigma_{\text{SSB-BLK}}^{n_{\text{SSB-BLK}}}$, and $\pi_{\text{SSB}} \xleftarrow{\$} \mathsf{SSB.Open}(\text{HK}, x, i)$, we have $\mathsf{SSB.Verify}(\text{HK}, \mathcal{H}_{\text{HK}}(x), i, x_i, \pi_{\text{SSB}}) = 1$.

► **Index Hiding**: The index hiding property of an SSB hash is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ chooses an integer $n_{\text{SSB-BLK}} \leq 2^\lambda$ together with a pair of indices $i_0^*, i_1^* \in [0, n_{\text{SSB-BLK}} - 1]$, and submits them to $\mathcal{C}$.
- $\mathcal{C}$ selects a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$ and computes $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}}, i_{\hat{b}}^*)$, and returns HK to $\mathcal{B}$.
- $\mathcal{B}$ eventually outputs a guess bit $\hat{b}' \in \{0,1\}$.

The SSB hash is said to be index hiding if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SSB,IH}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Somewhere Statistically Binding**: An SSB hash key HK is said to be statistically binding for an index $i^* \in [0, n_{\text{SSB-BLK}} - 1]$ if there do not exist any $h \in \{0,1\}^{\ell_{\text{SSB-HASH}}}, u \neq u' \in \Sigma_{\text{SSB-BLK}}$, and $\pi_{\text{SSB}}, \pi'_{\text{SSB}} \in \Pi_{\text{SSB}}$ such that $\mathsf{SSB.Verify}(\text{HK}, h, i^*, u, \pi_{\text{SSB}}) = 1 = \mathsf{SSB.Verify}(\text{HK}, h, i^*, u', \pi'_{\text{SSB}})$.

The SSB hash is defined to be somewhere statistically binding if for any security parameter λ, integer $n_{\text{SSB-BLK}} \leq 2^\lambda$, index $i^* \in [0, n_{\text{SSB-BLK}} - 1]$, the hash key $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}}, i^*)$ is statistically binding for i^*. Note that this is an information theoretic property.

The first construction of an SSB hash is presented by Hubacek et al. [15]. Their construction is based on fully homomorphic encryption (FHE) [**?**]. Recently, Okamoto et al. [25] provides alternative constructions of SSB hash based on various standard number theoretic assumptions. Such as the Decisional Diffie-Hellman assumption. In this paper, we consider $\ell_{\text{SSB-BLK}} = 1$ and $n_{\text{SSB-BLK}} = 2^\lambda$.

2.3.3 Positional Accumulator

We will now present the notion of a positional accumulator as defined by Koppula et al. [21]. Intuitively, a positional accumulator is a cryptographic data structure that maintains two values, namely, a storage value and an accumulator value. The storage value is allowed to grow comparatively large, while the accumulator value is constrained to be short. Message symbols can be written to various positions in the

underlying storage, and new accumulated values can be computed as a string, knowing only the previous accumulator value and the newly written symbol together with 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, there are additional helper algorithms which essentially allow a party who is maintaining the full storage to help a more restricted party maintaining only the accumulator value recover the data currently written at an arbitrary location. The helper is not necessarily trusted, so the party maintaining the accumulator value performs a verification procedure in order to be convinced that it is indeed reading the correct symbols.

Definition 2.4 (Positional Accumulator [21]). A positional accumulator consists of the PPT algorithms (ACC.Setup, ACC.Setup-Enforce-Read, ACC.Setup-Enforce-Write, ACC.Prep-Read, ACC.Prep-Write, ACC.Verify-Read, ACC.Write-Store, ACC.Update) along with a block alphabet $\Sigma_{\text{ACC-BLK}} = \{0,1\}^{\ell_{\text{ACC-BLK}}}$, accumulator size $\ell_{\text{ACC-ACCUMULATE}}$, proof space $\Pi_{\text{ACC}} = \{0,1\}^{\ell_{\text{ACC-PROOF}}}$ where $\ell_{\text{ACC-BLK}}, \ell_{\text{ACC-ACCUMULATE}}$, $\ell_{\text{ACC-PROOF}}$ are some polynomials in the security parameter λ. The algorithms have the following syntax:

- $\mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}}) \to (\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$: The setup authority takes as input the security parameter 1^λ and an integer $n_{\text{ACC-BLK}} \leq 2^\lambda$ representing the maximum number of blocks that can be accumulated. It outputs the public parameters PP_{ACC}, an initial accumulator value w_0, and an initial storage value STORE_0.
- $\mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1,i_1),\ldots,(x_\kappa,i_\kappa)), i^*) \to (\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$: Taking as input the security parameter 1^λ, an integer $n_{\text{ACC-BLK}} \leq 2^\lambda$ representing the maximum number of blocks that can be accumulated, a sequence of symbol-index pairs $((x_1,i_1),\ldots,(x_\kappa,i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, and an additional index $i^* \in [0, n_{\text{ACC-BLK}} - 1]$, the setup authority publishes the public parameters PP_{ACC}, an initial accumulator value w_0, together with an initial storage value STORE_0.
- $\mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1,i_1),\ldots,x_\kappa,i_\kappa))) \to (\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$: On input the security parameter 1^λ, an integer $n_{\text{ACC-BLK}} \leq 2^\lambda$ denoting the maximum number of blocks that can be accumulated, and a sequence of symbol-index pairs $((x_1,i_1),\ldots,(x_\kappa,i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, the setup authority publishes the public parameters PP_{ACC}, an initial accumulator value w_0, as well as, an initial storage value STORE_0.
- $\mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_{\text{IN}}, i_{\text{IN}}) \to (x_{\text{OUT}}, \pi_{\text{ACC}})$: A storage-maintaining party takes as input the public parameter PP_{ACC}, a storage value STORE_{IN}, and an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$. It outputs a symbol $x_{\text{OUT}} \in \Sigma_{\text{ACC-BLK}} \cup \{\epsilon\}$ (ϵ being the empty string) and a proof $\pi_{\text{ACC}} \in \Pi_{\text{ACC}}$.
- $\mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{\text{IN}}, i_{\text{IN}}) \to \text{AUX}$: Taking as input the public parameter PP_{ACC}, a storage value STORE_{IN}, together with an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$, a storage-maintaining party outputs an auxiliary value AUX.
- $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, x_{\text{IN}}, i_{\text{IN}}, \pi_{\text{ACC}}) \to \hat{\beta} \in \{0,1\}$: A verifier takes as input the public parameter PP_{ACC}, an accumulator value $w_{\text{IN}} \in \{0,1\}^{\ell_{\text{ACC-ACCUMULATE}}}$, a symbol $x_{\text{IN}} \in \Sigma_{\text{ACC-BLK}} \cup \{\epsilon\}$, an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$, and a proof $\pi_{\text{ACC}} \in \Pi_{\text{ACC}}$. It outputs a bit $\hat{\beta} \in \{0,1\}$.
- $\mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{\text{IN}}, i_{\text{IN}}, x_{\text{IN}}) \to \text{STORE}_{\text{OUT}}$: On input the public parameters PP_{ACC}, a storage value STORE_{IN}, an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$, and a symbol $x_{\text{IN}} \in \Sigma_{\text{ACC-BLK}}$, a storage-maintaining party computes a new storage value $\text{STORE}_{\text{OUT}}$.
- $\mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, x_{\text{IN}}, i_{\text{IN}}, \text{AUX}) \to w_{\text{OUT}} \text{or} \perp$: An accumulator-updating party takes as input the public parameters PP_{ACC}, an accumulator value $w_{\text{IN}} \in \{0,1\}^{\ell_{\text{ACC-ACCUMULATE}}}$, a symbol $x_{\text{IN}} \in \Sigma_{\text{ACC-BLK}}$, an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$, and an auxiliary value AUX. It outputs the updated accumulator value $w_{\text{OUT}} \in \{0,1\}^{\ell_{\text{ACC-ACCUMULATE}}}$ or the designated reject string $\perp$.

Following [21, 10], in this paper we will consider the algorithms ACC.Setup, ACC.Setup-Enforce-Read, and ACC.Setup-Enforce-Write as randomized while all other algorithms as deterministic.

► **Correctness**: Consider any symbol-index pair sequence $((x_1,i_1),\ldots,(x_\kappa,i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$. Fix any $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$. For $j = 1,\ldots,\kappa$, iteratively define the following:

- $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j, x_j)$
- $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j)$
- $w_j = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{j-1}, x_j, i_j, \text{AUX}_j)$

The following correctness properties are required to be satisfied:

i) For any security parameter λ, $n_{\text{ACC-BLK}} \leq 2^\lambda$, index $i^* \in [0, n_{\text{ACC-BLK}}-1]$, sequence of symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}}-1])^\kappa$, and $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$, if STORE_κ is computed as above, then $\mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_\kappa, i^*)$ returns (x_j, π_{ACC}) where j is the largest value in $[\kappa]$ such that $i_j = i^*$.

ii) For any security parameter λ, $n_{\text{ACC-BLK}} \leq 2^\lambda$, sequence of symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}}-1])^\kappa$, $i^* \in [0, n_{\text{ACC-BLK}}-1]$, and $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$, if STORE_κ and w_κ are computed as above and $(x_{\text{OUT}}, \pi_{\text{ACC}}) = \mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_\kappa, i^*)$, then $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_\kappa, x_{\text{OUT}}, i^*, \pi_{\text{ACC}}) = 1$

► **Indistinguishability of Read Setup**: This property of a positional accumulator is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ chooses a bound $n_{\text{ACC-BLK}} \leq 2^\lambda$ of the number of blocks, κ symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, and an index $i^* \in [0, n_{\text{ACC-BLK}} - 1]$. It submits all of those to $\mathcal{C}$.
- $\mathcal{C}$ selects a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, $\mathcal{C}$ generates $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$. Otherwise, $\mathcal{C}$ forms $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)), i^*)$. It returns $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The positional accumulator is said to satisfy indistinguishability of read setup if for any PPT adversary $\mathcal{B}$, for any security parameter λ, we have

$$\mathsf{Adv}_{\mathcal{B}}^{\text{ACC,IND-READ}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Indistinguishability of Write Setup**: This property of a positional accumulator is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ chooses a bound $n_{\text{ACC-BLK}} \leq 2^\lambda$ of the number of blocks, and κ symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$. It submits all of those to $\mathcal{C}$.
- $\mathcal{C}$ selects a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, $\mathcal{C}$ generates $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$. Otherwise, $\mathcal{C}$ generates $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)))$. It returns $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

A positional accumulator is said to satisfy indistinguishability of write setup if for any PPT adversary $\mathcal{B}$, for any security parameter λ, we have

$$\mathsf{Adv}_{\mathcal{B}}^{\text{ACC,IND-WRITE}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Read Enforcing**: Consider any security parameter λ, $n_{\text{ACC-BLK}} \leq 2^\lambda$, $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}}-1])^\kappa$, and $i^* \in [0, n_{\text{ACC-BLK}}-1]$. Let $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)), i^*)$. For $j = 1, \ldots, \kappa$, iteratively define the following:

- $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j, x_j)$
- $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j)$
- $w_j = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{j-1}, x_j, i_j, \text{AUX}_j)$

The positional accumulator is said to be read enforcing if $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_\kappa, x_{\text{IN}}, i^*, \pi_{\text{ACC}}) = 1$ implies either $[i^* \notin \{i_1, \ldots, i_\kappa\}] \wedge [x_{\text{IN}} = \epsilon]$ or $x_{\text{IN}} = x_j$ for the largest $j \in [\kappa]$ such that $i_j = i^*$. Note that this is an information theoretic property.

► **Write Enforcing**: Consider any security parameter λ, $n_{\text{ACC-BLK}} \leq 2^\lambda$, and $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}}-1])^\kappa$. Let $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)))$. For $j = 1, \ldots, \kappa$, iteratively define the following:

- $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j, x_j)$
- $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j)$

– $w_j = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{j-1}, x_j, i_j, \mathrm{AUX}_j)$

The positional accumulator is defined to be write enforcing if $\mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\kappa-1}, x_\kappa, i_\kappa, \mathrm{AUX}) = w_{\mathrm{OUT}} \neq \perp$, for any AUX, implies $w_{\mathrm{OUT}} = w_\kappa$. Observe that this is an information theoretic property as well.

The first construction of a positional accumulator is presented by Koppula et al. [21] based on IO and one-way function. Recently, Okamoto et al. [25] provided an alternative construction of positional accumulator from standard number theoretic assumptions. Such as the Decisional Diffie-Hellman assumption.

2.3.4 Iterator

Here, we define cryptographic iterators again following [21]. Informally speaking, a cryptographic iterator consists of a small state that is updated in an iterative fashion as messages are received. An update to incorporate a new message given the current state is performed with the help of some public parameters. Since, states are relatively small regardless of the number of messages that have been iteratively incorporated, there is in general many sequences of messages that lead to the same state. However, the security property requires that the normal public parameters should be computationally indistinguishable from specially constructed enforcing parameters which ensure that a particular state can only be obtained as the outcome of an update to precisely one other state-message pair. Note that this enforcement is a very localized property to a specific state and hence can be achieved information-theoretically when it is fixed ahead of time where exactly this enforcement is desired.

Definition 2.5 (Iterator [21]). A cryptographic iterator consists of the PPT algorithms (ITR.Setup, ITR.Setup-Enforce, ITR.Iterate) along with a message space $\mathcal{M}_{\mathrm{ITR}} = \{0,1\}^{\ell_{\text{ITR-MSG}}}$ and iterator state size $\ell_{\text{ITR-ST}}$, where $\ell_{\text{ITR-MSG}}, \ell_{\text{ITR-ST}}$ are some polynomials in the security parameter λ. Algorithms have the following syntax:

- $\mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}}) \to (\mathrm{PP}_{\mathrm{ITR}}, v_0)$: The setup authority takes as input the security parameter 1^λ along with an integer bound $n_{\mathrm{ITR}} \leq 2^\lambda$ on the number of iterations. It outputs the public parameters $\mathrm{PP}_{\mathrm{ITR}}$ and an initial state $v_0 \in \{0,1\}^{\ell_{\text{ITR-ST}}}$.
- $\mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\mathrm{ITR}}, (\mu_1, \ldots, \mu_\kappa)) \to (\mathrm{PP}_{\mathrm{ITR}}, v_0)$: Taking as input the security parameter 1^λ, an integer bound $n_{\mathrm{ITR}} \leq 2^\lambda$, together with a sequence of κ messages $(\mu_1, \ldots, \mu_\kappa) \in \mathcal{M}_{\mathrm{ITR}}^\kappa$, where $\kappa \leq n_{\mathrm{ITR}}$, the setup authority publishes the public parameters $\mathrm{PP}_{\mathrm{ITR}}$ and an initial state $v_0 \in \{0,1\}^{\ell_{\text{ITR-ST}}}$.
- $\mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}} \in \{0,1\}^{\ell_{\text{ITR-ST}}}, \mu) \to v_{\mathrm{OUT}}$: On input the public parameters $\mathrm{PP}_{\mathrm{ITR}}$, a state v_{IN}, and a message $\mu \in \mathcal{M}_{\mathrm{ITR}}$, an iterator outputs an updated state $v_{\mathrm{OUT}} \in \{0,1\}^{\ell_{\text{ITR-ST}}}$. For any integer $\kappa \leq n_{\mathrm{ITR}}$, we will write $\mathsf{ITR.Iterate}^\kappa(\mathrm{PP}_{\mathrm{ITR}}, v_0, (\mu_1, \ldots, \mu_\kappa))$ to denote $\mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\kappa-1}, \mu_\kappa)$, where v_j is defined iteratively as $v_j = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{j-1}, \mu_j)$ for all $j = 1, \ldots, \kappa - 1$.

The algorithm ITR.Iterate is deterministic, while the other two are randomized.

► **Indistinguishability of Enforcing Setup**: This property of a cryptographic iterator is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ chooses an integer bound $n_{\mathrm{ITR}} \leq 2^\lambda$, along with a sequence of κ messages $(\mu_1, \ldots, \mu_\kappa) \in \mathcal{M}_{\mathrm{ITR}}^\kappa$, and submits them to $\mathcal{C}$.
- $\mathcal{C}$ selects a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, $\mathcal{C}$ generates $(\mathrm{PP}_{\mathrm{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}})$. Else, $\mathcal{C}$ generates $(\mathrm{PP}_{\mathrm{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\mathrm{ITR}}, (\mu_1, \ldots, \mu_\kappa))$. It sends back $(\mathrm{PP}_{\mathrm{ITR}}, v_0)$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The cryptographic iterator is said to satisfy indistinguishability of enforcing setup if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{ITR,IND-ENF}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Enforcing**: Consider any security parameter λ, $n_{\mathrm{ITR}} \leq 2^\lambda, \kappa \leq n_{\mathrm{ITR}}$, and $(\mu_1, \ldots, \mu_\kappa) \in \mathcal{M}_{\mathrm{ITR}}^\kappa$. Let $(\mathrm{PP}_{\mathrm{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Set\text{-}Enforce}(1^\lambda, n_{\mathrm{ITR}}, (\mu_1, \ldots, \mu_\kappa))$ and $v_j = \mathsf{ITR.Iterate}^j(\mathrm{PP}_{\mathrm{ITR}}, v_0, (\mu_1, \ldots, \mu_j))$ for all $j \in [\kappa]$. The cryptographic iterator is said to be enforcing if $v_k = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v', \mu')$ implies $(v', \mu') = (v_{\kappa-1}, \mu_\kappa)$. Note that this is an information theoretic property.

Koppula et al. [21] have presented a construction of cryptographic iterators from IO and one-way function.

2.3.5 Splittable Signature

The following background on splittable signatures is taken verbatim from Koppula et al. [21] as well. A splittable signature scheme is essentially a normal signature scheme augmented by some additional algorithms that produce alternative signing and verification keys with different capabilities. More precisely, there are "all-but-one" signing and verification keys which work correctly for all messages except for a specific one, as well as there are "one" signing and verification keys which work only for a particular message. Additionally, there are "reject" verification keys which always reject signatures.

Definition 2.6 (Splittable Signature: SPS [21]). A splittable signature scheme (SPS) for message space $\mathcal{M}_{\text{SPS}} = \{0,1\}^{\ell_{\text{SPS-MSG}}}$ and signature space $\mathcal{S}_{\text{SPS}} = \{0,1\}^{\ell_{\text{SPS-SIG}}}$, where $\ell_{\text{SPS-MSG}}, \ell_{\text{SPS-SIG}}$ are some polynomials in the security parameter λ, consists of the PPT algorithms (SPS.Setup, SPS.Sign, SPS.Verify, SPS.Split, SPS.Sign-ABO) which are described below:

- $\mathsf{SPS.Setup}(1^\lambda) \rightarrow (\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}})$: The setup authority takes as input the security parameter 1^λ and generates a signing key SK_{SPS}, a verification key VK_{SPS}, together with a reject verification key $\text{VK}_{\text{SPS-REJ}}$.
- $\mathsf{SPS.Sign}(\text{SK}_{\text{SPS}}, m) \rightarrow \sigma_{\text{SPS}}$: A signer given a signing key SK_{SPS} along with a message $m \in \mathcal{M}_{\text{SPS}}$, produces a signature $\sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}$.
- $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS}}, m, \sigma_{\text{SPS}}) \rightarrow \hat{\beta} \in \{0,1\}$: A verifier takes as input a verification key VK_{SPS}, a message $m \in \mathcal{M}_{\text{SPS}}$, and a signature $\sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}$. It outputs a bit $\hat{\beta} \in \{0,1\}$.
- $\mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*) \rightarrow (\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$: On input a signing key SK_{SPS} along with a message $m^* \in \mathcal{M}_{\text{SPS}}$, the setup authority generates a signature $\sigma_{\text{SPS-ONE},m^*} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS}}, m^*)$, a one-message verification key $\text{VK}_{\text{SPS-ONE}}$, and all-but-one signing-verification key pair $(\text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$.
- $\mathsf{SPS.Sign\text{-}ABO}(\text{SK}_{\text{SPS-ABO}}, m) \rightarrow \sigma_{\text{SPS}}$ or $\perp$: An all-but-one signer given an all-but-one signing key $\text{SK}_{\text{SPS-ABO}}$ and a message $m \in \mathcal{M}_{\text{SPS}}$, outputs a signature $\sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}$ or a distinguished string $\perp$ to indicate failure. For simplicity of notation, we will often use $\mathsf{SPS.Sign}(\text{SK}_{\text{SPS-ABO}}, m)$ to represent the output of this algorithm.

We note that among the algorithms described above, SPS.Setup and SPS.Split are randomized while all the others are deterministic.

► **Correctness**: For any security parameter λ, message $m^* \in \mathcal{M}_{\text{SPS}}$, $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, and $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*)$ the following correctness conditions hold:

i) $\forall\, m \in \mathcal{M}_{\text{SPS}},\ \mathsf{SPS.Verify}(\text{VK}_{\text{SPS}}, m, \mathsf{SPS.Sign}(\text{SK}_{\text{SPS}}, m)) = 1.$
ii) $\forall\, m \neq m^* \in \mathcal{M}_{\text{SPS}},\ \mathsf{SPS.Sign}(\text{SK}_{\text{SPS}}, m) = \mathsf{SPS.Sign\text{-}ABO}(\text{SK}_{\text{SPS-ABO}}, m).$
iii) $\forall\, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}},\ \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-ONE}}, m^*, \sigma_{\text{SPS}}) = \mathsf{SPS.Verify}(\text{VK}_{\text{SPS}}, m^*, \sigma_{\text{SPS}}).$
iv) $\forall\, m \neq m^* \in \mathcal{M}_{\text{SPS}}, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}},\ \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-ABO}}, m, \sigma_{\text{SPS}}) = \mathsf{SPS.Verify}(\text{VK}_{\text{SPS}}, m, \sigma_{\text{SPS}}).$
v) $\forall\, m \neq m^* \in \mathcal{M}_{\text{SPS}}, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}},\ \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-ONE}}, m, \sigma_{\text{SPS}}) = 0.$
vi) $\forall\, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}},\ \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-ABO}}, m^*, \sigma_{\text{SPS}}) = 0.$
vii) $\forall\, m \in \mathcal{M}_{\text{SPS}}, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}},\ \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-REJ}}, m, \sigma_{\text{SPS}}) = 0.$

► $\text{VK}_{\text{SPS-REJ}}$ **Indistinguishability**: This property of a splittable signature scheme is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{C}$ generates $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$. Next it chooses a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, it sends VK_{SPS} to $\mathcal{B}$. Otherwise, it sends $\text{VK}_{\text{SPS-REJ}}$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The splittable signature scheme is said to be $\text{VK}_{\text{SPS-REJ}}$ indistinguishable if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SPS,IND-REJ}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► $\text{VK}_{\text{SPS-ONE}}$ **Indistinguishability**: This feature of a splittable signature scheme is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ submits a message $m^* \in \mathcal{M}_{\text{SPS}}$ to $\mathcal{C}$.
- $\mathcal{C}$ generates $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$. Next it computes $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*)$. Then it chooses a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, it returns $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}})$ to $\mathcal{B}$. Else, it returns $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS}})$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The splittable signature scheme is said to be $\text{VK}_{\text{SPS-ONE}}$ indistinguishable if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SPS,IND-ONE}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **$\text{VK}_{\text{SPS-ABO}}$ Indistinguishability**: This feature of a splittable signature scheme is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ submits a message $m^* \in \mathcal{M}_{\text{SPS}}$ to $\mathcal{C}$.
- $\mathcal{C}$ generates $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$. Next it computes $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*)$. Then it chooses a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, it returns $(\text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$ to $\mathcal{B}$. Else, it returns $(\text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS}})$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The splittable signature scheme is said to be $\text{VK}_{\text{SPS-ABO}}$ indistinguishable if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SPS,IND-ABO}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Splitting Indistinguishability**: This feature of a splittable signature scheme is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ submits a message $m^* \in \mathcal{M}_{\text{SPS}}$ to $\mathcal{C}$.
- $\mathcal{C}$ forms $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda), (\text{SK}'_{\text{SPS}}, \text{VK}'_{\text{SPS}}, \text{VK}'_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$. Next it computes $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*)$ as well as $(\sigma'_{\text{SPS-ONE},m^*}, \text{VK}'_{\text{SPS-ONE}}, \text{SK}'_{\text{SPS-ABO}}, \text{VK}'_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}'_{\text{SPS}}, m^*)$. Then it chooses a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, it returns $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$ to $\mathcal{B}$. Else, it returns $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}'_{\text{SPS-ABO}}, \text{VK}'_{\text{SPS-ABO}})$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The splittable signature scheme is said to be splitting indistinguishable if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SPS,IND-SPL}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Koppula et al. [21] have constructed a splittable signature scheme using IO and one-way function.

3 Our Attribute-Based Signature for Turing Machines

3.1 Notion

Here we will formally define the notion of an attribute-based signature scheme where signing policies are associated with TM's. This definition is similar to that defined in [30, 28] for circuits. However, due to the use of TM's as opposed to circuits, such a scheme can handle signing attribute strings of arbitrary polynomial length.

Definition 3.1 (Attribute-Based Signature for Turing Machines: ABS). Let $\mathbb{M}_\lambda$ be a class of TM's, the members of which have (worst-case) running time bounded by $T = 2^\lambda$. An attribute-based signature (ABS) scheme for signing policies associated with the TM's in $\mathbb{M}_\lambda$ comprises of an attribute universe $\mathcal{U}_{\text{ABS}} \subset \{0,1\}^*$, a message space $\mathcal{M}_{\text{ABS}} = \{0,1\}^{\ell_{\text{ABS-MSG}}}$, a signature space $\mathcal{S}_{\text{ABS}} = \{0,1\}^{\ell_{\text{ABS-SIG}}}$, where $\ell_{\text{ABS-MSG}}, \ell_{\text{ABS-SIG}}$ are some polynomials in the security parameter λ, and the PPT algorithms (ABS.Setup, ABS.KeyGen, ABS.Sign, ABS.Verify) described below:

- $\mathsf{ABS.Setup}(1^\lambda) \rightarrow (\text{PP}_{\text{ABS}}, \text{MSK}_{\text{ABS}})$: The setup authority takes as input the security parameter 1^λ. It publishes the public parameters PP_{ABS} while generates a master secret key MSK_{ABS} for itself.
- $\mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M) \rightarrow \text{SK}_{\text{ABS}}(M)$: Taking as input the master secret key MSK_{ABS} and a signing policy TM $M \in \mathbb{M}_\lambda$ of a signer, the setup authority provides the corresponding signing key $\text{SK}_{\text{ABS}}(M)$ to the legitimate signer.
- $\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg}) \rightarrow \sigma_{\text{ABS}}$ or $\perp$: On input the signing key $\text{SK}_{\text{ABS}}(M)$ corresponding to the legitimate signing policy TM $M \in \mathbb{M}_\lambda$, a signing attribute string $x \in \mathcal{U}_{\text{ABS}}$, and a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, a signer outputs either a signature $\sigma_{\text{ABS}} \in \mathcal{S}_{\text{ABS}}$ or $\perp$ indicating failure.
- $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x, \mathsf{msg}, \sigma_{\text{ABS}}) \rightarrow \hat{\beta} \in \{0,1\}$: A verifier takes as input the public parameters PP_{ABS}, a signing attribute string $x \in \mathcal{U}_{\text{ABS}}$, a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, and a purported signature $\sigma_{\text{ABS}} \in \mathcal{S}_{\text{ABS}}$. It outputs a bit $\hat{\beta} \in \{0,1\}$.

We note that all the algorithms described above except ABS.Verify are randomized. The algorithms satisfy the following properties:

► **Correctness**: For any security parameter λ, $(\text{PP}_{\text{ABS}}, \text{MSK}_{\text{ABS}}) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$, $M \in \mathbb{M}_\lambda$, $\text{SK}_{\text{ABS}}(M) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M)$, $x \in \mathcal{U}_{\text{ABS}}$, and $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, if $M(x) = 1$, then $\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg})$ outputs $\sigma_{\text{ABS}} \in \mathcal{S}_{\text{ABS}}$ such that $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x, \mathsf{msg}, \sigma_{\text{ABS}}) = 1$.

► **Signer Privacy**: An ABS scheme is said to provide signer privacy if for any security parameter λ, message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, $(\text{PP}_{\text{ABS}}, \text{MSK}_{\text{ABS}}) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$, signing policies $M, M' \in \mathbb{M}_\lambda$, signing keys $\text{SK}_{\text{ABS}}(M) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M), \text{SK}_{\text{ABS}}(M') \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M')$, $x \in \mathcal{U}_{\text{ABS}}$ such that $M(x) = 1 = M'(x)$, the distributions of the signatures outputted by $\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg})$ and $\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M'), x, \mathsf{msg})$ are identical.

► **Existential Unforgeability against Selective Attribute Adaptive Chosen Message Attack**: This property of an ABS scheme is defined through the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$:

- $\mathcal{A}$ submits a challenge attribute string $x^* \in \mathcal{U}_{\text{ABS}}$ to $\mathcal{B}$.
- $\mathcal{B}$ generates $(\text{PP}_{\text{ABS}}, \text{MSK}_{\text{ABS}}) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$ and provides $\mathcal{A}$ with PP_{ABS}.
- $\mathcal{A}$ may adaptively make a polynomial number of queries of the following types:
  - **Signing key query**: When $\mathcal{A}$ queries a signing key corresponding to a signing policy TM $M \in \mathbb{M}_\lambda$ subject to the restriction that $M(x^*) = 0$, $\mathcal{B}$ gives back $\text{SK}_{\text{ABS}}(M) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M)$ to $\mathcal{A}$.
  - **Signature query**: When $\mathcal{A}$ queries a signature on a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$ under an attribute string $x \in \mathcal{U}_{\text{ABS}}$, $\mathcal{B}$ samples a signing policy TM $M \in \mathbb{M}_\lambda$ such that $M(x) = 1$, creates a signing key $\text{SK}_{\text{ABS}}(M) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M)$, and generates a signature $\sigma_{\text{ABS}} \xleftarrow{\$} \mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg})$, which $\mathcal{B}$ returns to $\mathcal{A}$.
- At the end of interaction $\mathcal{A}$ outputs a message-signature pair $(\mathsf{msg}^*, \sigma^*_{\text{ABS}})$. $\mathcal{A}$ wins if the following hold simultaneously:
  i) $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\text{ABS}}) = 1$.
  ii) $\mathcal{A}$ has not made any signature query on msg^* under x^*.

The ABS scheme is said to be existentially unforgeable against selective attribute adaptive chosen message attack if for any PPT adversary $\mathcal{A}$, for any security parameter λ,

$$\mathsf{Adv}^{\text{ABS,UF-CMA}}_{\mathcal{A}}(\lambda) = \mathsf{Pr}[\mathcal{A} \text{ wins}] \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Remark 3.1. Note that in the existential unforgeability experiment above without loss of generality, we can consider signature queries on messages only under the challenge attribute string x^*. This is because any signature query under some attribute string $x \neq x^*$ can be replaced by a signing key query for a signing policy TM $M_x \in \mathbb{M}_\lambda$ that accepts only the string x. Since $x \neq x^*$, $M_x(x^*) = 0$, and thus M_x forms a valid signing key query. We will use this simplification in our proof.

3.2 Principal Ideas behind Our ABS Scheme

Here we give an overview of our ABS scheme. To generate an ABS signature, we use a PPRF $\mathcal{F}$, an SSB hash function, and a standard existentially unforgeable digital signature scheme SIG for the same message space $\mathcal{M}_{\text{ABS}} = \{0,1\}^{\ell_{\text{ABS-MSG}}}$ of the ABS scheme. A high level description of the structure of our ABS signatures and the signature verification process is presented below:

a) The master signing key MSK_{ABS} of our ABS scheme consists of a key K for the PPRF $\mathcal{F}$ and an SSB hash key HK.
b) The ABS signature on a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$ under certain signing attribute string $x = x_0 \dots x_{\ell_x - 1} \in \mathcal{U}_{\text{ABS}} \subset \{0,1\}^*$ of length $|x| = \ell_x$ is $\sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}})$, which consists of a verification key VK_{SIG} and a signature σ_{SIG} of the digital signature scheme SIG, computed as follows:
The signing attribute string x is hashed using the SSB hash key HK to form a fixed length hash value h. Next, a pseudorandom string $r_{\text{SIG}} = \mathcal{F}(K, h)$ is computed using the PPRF $\mathcal{F}$. The setup algorithm of SIG is then executed with the randomness r_{SIG} to generate the SIG verification key VK_{SIG} along with its corresponding SIG signing key SK_{SIG}. The SIG signing key SK_{SIG} is utilized to generate the SIG signature σ_{SIG} on the message msg by running the SIG signing algorithm.
c) The public parameters PP_{ABS} corresponding to the master signing key MSK_{ABS} is comprised of the SSB hash key HKtogether with an IO-obfuscated program $\mathcal{V}_{\text{ABS}}$, known as the *verifying program* (see Fig. 3.1). The verifying program $\mathcal{V}_{\text{ABS}}$ has the PPRF key K hardwired in it, and takes as input an SSB hash value h. The program $\mathcal{V}_{\text{ABS}}$ generates the pseudorandom string $\hat{r}_{\text{SIG}} = \mathcal{F}(K, h)$, performs the SIG setup algorithm using $\hat{r}_{\text{SIG}}$ as the randomness, and outputs the resulting SIG verification key $\widehat{\text{VK}}_{\text{SIG}}$.
d) A verifier checks a purported signature $\sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}})$ on some message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$ under some signing attribute string $x = x_0 \dots x_{\ell_x - 1} \in \mathcal{U}_{\text{ABS}}$ with $|x| = \ell_x$ using the public parameters PP_{ABS} as follows:
It first hashes x using the SSB hash key HK forming the hash value h. Next, it obtains an SIG verification key $\widehat{\text{VK}}_{\text{SIG}}$ by running the obfuscated verifying program $\mathcal{V}_{\text{ABS}}$ on input the hash value h. The verifier accepts the signature σ_{ABS} if the SIG verification key $\widehat{\text{VK}}_{\text{SIG}}$ outputted by the program $\mathcal{V}_{\text{ABS}}$ matches VK_{SIG} and the SIG signature σ_{SIG} is verified for msg with respect to VK_{SIG}. The correctness of the verification process is immediate from the correctness of SIG along with the functional property of the PPRF $\mathcal{F}$ and the SSB hash function.

We now explain the structure of the ABS signing keys associated with specific signing policy TM's and justify the utility of the IO-obfuscated programs included in our ABS signing keys. The first intuition is to design the signing key corresponding to certain signing policy TM M in such a way that while attempting to sign any message with respect to some signing attribute string $x \in \mathcal{U}_{\text{ABS}}$, the signer is forced to execute M on x and succeeds in obtaining a valid signature if and only if M accepts x. We plan to make this idea work by providing an IO-obfuscated program $\mathcal{P}_{\text{ABS}}$, called the *next step program* (see Fig. 3.5). We sketch the functionality of $\mathcal{P}_{\text{ABS}}$ below:

i) The program $\mathcal{P}_{\text{ABS}}$ has the PPRF key K hardwired in it. It takes as input a state and header position of M, together with an input symbol and an SSB hash value. It computes the next step function of M on the input state-symbol pair. In case M enters the accepting state, the program outputs the desired SIG signing key-verification key pair as follows:
It first generates a pseudorandom string by applying the PPRF $\mathcal{F}$ with key K on the input SSB hash value. Next, it runs the SIG setup algorithm utilizing the generated pseudorandom string to produce the SIG signing key-verification key pair to be outputted. Observe that once the required SIG signing key-verification key is obtained, the signer can itself compute an SIG signature on any message.
ii) To prevent illegal inputs across successive invocations, the program $\mathcal{P}_{\text{ABS}}$ must perform certain authenticity checks before executing the next step function of M. A natural choice would be to maintain a short authenticated commitment of the current tape configuration of the TM M that is updated and re-authenticated at each invocation of $\mathcal{P}_{\text{ABS}}$. For this, we make use of a positional accumulator ACC and a splittable signature scheme SPS. Our security proof crucially relies on various features of splittable signatures in the hybrid transformations. We include a fresh set of public parameters PP_{ACC} of the positional accumulator within each ABS signing key. We design the program $\mathcal{P}_{\text{ABS}}$ to additionally take as input an accumulator value, a proof for the accumulator value, together with an SPS signature on the input state, header position, and the input accumulator value.
iii) The program $\mathcal{P}_{\text{ABS}}$ verifies the SPS signature and checks the accumulator proof to get convinced that the input symbol is indeed the one placed at the input header position of the underlying storage of the input accumulator value.

iv) If all these verifications pass, then $\mathcal{P}_{\text{ABS}}$ determines the next state and header position of M, as well as computes the new symbol that needs to be written to the input header position. The program $\mathcal{P}_{\text{ABS}}$ then updates the accumulator value by placing the new symbol at the input header position, and generates an SPS signature on the updated accumulator value along with the computed next state and header position of M. In order to deal with the positional accumulator related verifications and updations, $\mathcal{P}_{\text{ABS}}$ has PP_{ACC} hardwired.

Now, observe that before starting the repeated execution of $\mathcal{P}_{\text{ABS}}$, the signer would require an SPS signature on the initial accumulator value formed by accumulating the bits of the signing attribute string along with the initial state and header position of M. For this, we include another IO-obfuscated program $\mathcal{P}_1$, known as the *initial signing programm* (see Fig. 3.2). It takes as input an accumulator value and outputs an SPS signature on the tuple of the input accumulator value, initial state, and initial header position of M. The idea is that while signing some message under some signing attribute string x using a signing key corresponding to some TM M, the signer would proceed as follows:

a) It would first compute the SSB hash value h by hashing x using the system wide SSB hash key HK, which is part of the public parameters PP_{ABS}.
b) The signer would also compute the accumulator value w_{INP} by accumulating the bits of x using the public parameters PP_{ACC} of positional accumulator included in the ABS signing key.
c) Then, using the obfuscated initial signing program $\mathcal{P}_1$, included in the signing key, the signer would obtain an SPS signature on w_{INP} along with the initial state and header position of M.
d) Finally, the signer would repeatedly run the obfuscated next step program $\mathcal{P}_{\text{ABS}}$, included in the signing key, each time giving as input all the quantities as mentioned above. Note that the hash value that needs to be given as input to $\mathcal{P}_{\text{ABS}}$ in each iteration is h. In case $\mathcal{P}_{\text{ABS}}$ reaches the accepting state, it would require h to apply $\mathcal{F}$ for producing the pseudorandom string to be used in the SIG setup algorithm.

However, this approach is not completely sound yet. Observe that, a possibly malicious signer can compute the SSB hash value h on the signing attribute string x, under which it wishes to produce the ABS signature, although M does not accepts it, and initiates the signing process by accumulating the bits of only a substring of x or some entirely different signing attribute string, which is accepted by M. To prevent such malicious behavior, we include another IO-obfuscated program $\mathcal{P}_2$ within the ABS signing key, known as the *accumulating program* (see Fig. 3.3), whose purpose is to *restrict* the signer from accumulating the bits of a different signing input string rather than the hashed one. The program $\mathcal{P}_2$ functions as follows:

i) The program $\mathcal{P}_2$ takes as input an SSB hash value h, an index i, a symbol, an accumulator value, an SPS signature on the input accumulator value (along with the initial state and header position of M), and an opening value for SSB.
ii) The program $\mathcal{P}_2$ verifies the SPS signature. Using the input opening value for SSB, it also checks whether the input symbol is indeed present at the index i of the string that has been hashed to form h, using the input opening value.
iii) If all of these verifications pass, then $\mathcal{P}_2$ updates the input accumulator value by writing the input symbol at the i^{th} position of the accumulator storage.

We also modify the obfuscated initial signing program $\mathcal{P}_1$, included in the signing key, to take as input an SSB hash value rather than an accumulator value and output an SPS signature on the accumulator value corresponding to the empty accumulator storage, along with the initial state and header position of M.

To forbid the signer from performing the signature generation by accumulating an M-accepted substring of the hashed signing attribute string, we generate the pseudorandom string to be used in the SIG setup algorithm by evaluating $\mathcal{F}$ on hash value-length pair of the signing attribute string instead of the hash value only. Without loss of generality, we set the upper bound of the length of signing attribute strings to be 2^λ, where λ is the underlying security parameter. Note that by suitably choosing λ, we can accommodate signing attribute strings of any polynomial length. Now, as the length of the signing attribute strings is bounded by 2^λ, it can be expressed as a bit strings of length λ. Thus, the signing attribute string length can be safely fed to $\mathcal{F}$ along with the SSB hash value of the signing attribute string. Hence, the obfuscated next step programs $\mathcal{P}_{\text{ABS}}$ included in our signing keys must also take as input the length of the attribute string for generating the pseudorandom string for the SIG setup algorithm if reaching to the accepting state. For the same reason, the verifying program $\mathcal{V}_{\text{ABS}}$ included in the public

parameters PP_{ABS} also needs to be modified to take as input the length of signing attribute strings along with their SSB hash values.

Therefore, in our ABS scheme, to generate an ABS signature on some message under certain signing attribute string using a signing key, corresponding to some TM M, a signer does the following:

a) It first hashes the signing attribute string.
b) It also obtains an SPS signature on the empty accumulator value included in the signing key, by running the obfuscated initial signing program $\mathcal{P}_1$ on input the computed hash value.
c) Next, it repeatedly runs the obfuscated accumulating program $\mathcal{P}_2$ to accumulate the bits of the signing attribute string.
d) It then runs the obfuscated next step program $\mathcal{P}_{\text{ABS}}$ iteratively on the current accumulator value along with other legitimate inputs until it obtains either the desired SIG signing key-verification key pair or $\perp$.
e) Finally, it uses the obtained SIG signing key to generate an SIG signature on the message it wishes to sign and outputs the pair of the SIG verification key obtained from $\mathcal{P}_{\text{ABS}}$ and the computed SIG signature on the message as the purported ABS signature.

To cope up with certain issues in the security proof we further include another IO-obfuscated program $\mathcal{P}_3$ in our ABS signing keys, known as the *signature changing program* (see Fig. 3.4). It changes the SPS signature on the accumulation of the bits of the signing attribute string before starting the iterative computation with the obfuscated next step program $\mathcal{P}_{\text{ABS}}$.

3.3 Formal Description of our ABS Construction

Let λ be the underlying security parameter. Let $\mathbb{M}_\lambda$ denote a family of TM's, the members of which have (worst case) running time bounded by $T = 2^\lambda$, input alphabet $\Sigma_{\text{INP}} = \{0,1\}$, and tape alphabet $\Sigma_{\text{TAPE}} = \{0,1,\_\}$. Our ABS construction supporting signing attribute universe $\mathcal{U}_{\text{ABS}} \subset \{0,1\}^*$, signing policies representable by TM's in $\mathbb{M}_\lambda$, and message space $\mathcal{M}_{\text{ABS}} = \{0,1\}^{\ell_{\text{ABS-MSG}}}$ utilizes the following cryptographic building blocks:

i) $\mathcal{IO}$: An indistinguishability obfuscator for general polynomial-size circuits.
ii) SSB = (SSB.Gen, $\mathcal{H}$, SSB.Open, SSB.Verify): A somewhere statistically binding hash function with $\Sigma_{\text{SSB-BLK}} = \{0,1\}$.
iii) ACC = (ACC.Setup, ACC.Setup-Enforce-Read, ACC.Setup-Enforce-Write, ACC.Prep-Read, ACC.Prep-Write, ACC.Verify-Read, ACC.Write-Store, ACC.Update): A positional accumulator with $\Sigma_{\text{ACC-BLK}} = \{0,1,\_\}$.
iv) ITR = (ITR.Setup, ITR.Setup-Enforce, ITR.Iterate): A cryptographic iterator with an appropriate message space $\mathcal{M}_{\text{ITR}}$
v) SPS = (SPS.Setup, SPS.Sign, SPS.Verify, SPS.Split, SPS.Sign-ABO): A splittable signature scheme with an appropriate message space $\mathcal{M}_{\text{SPS}}$.
vi) PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda}$: A length-doubling pseudorandom generator.
vii) $\mathcal{F}$ = ($\mathcal{F}$.Setup, $\mathcal{F}$.Puncture, $\mathcal{F}$.Eval): A puncturable pseudorandom function whose domain and range are chosen appropriately. For simplicity, we assume that $\mathcal{F}$ has inputs and outputs of bounded length instead of fixed length inputs and outputs. This assumption can be easily removed by using different PPRF's for different input and output lengths.
viii) SIG = (SIG.Setup, SIG.Sign, SIG.Verify): A digital signature scheme with associated message space $\mathcal{M}_{\text{ABS}} = \{0,1\}^{\ell_{\text{ABS}}}$ that is existentially unforgeable against chosen message attack (CMA).

The formal description of our ABS construction follows:

ABS.Setup$(1^\lambda) \to (\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{V}_{\text{ABS}}), \text{MSK}_{\text{ABS}} = (K, \text{HK}))$: The setup authority takes as input the security parameter 1^λ and proceeds as follows:

1. It first chooses a PPRF key $K \xleftarrow{\$} \mathcal{F}.\text{Setup}(1^\lambda)$.
2. Next it generates $\text{HK} \xleftarrow{\$} \text{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$.
3. After that, it forms the obfuscated program $\mathcal{V}_{\text{ABS}} = \mathcal{IO}(\text{Verify.Prog}_{\text{ABS}}[K])$, where the program $\text{Verify.Prog}_{\text{ABS}}$ is described in Fig. 3.1.
4. It keeps the master secret key $\text{MSK}_{\text{ABS}} = (K, \text{HK})$ and publishes the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{V}_{\text{ABS}})$.

ABS.KeyGen$(\text{MSK}_{\text{ABS}}, M) \to \text{SK}_{\text{ABS}}(M) = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{ABS}})$: On input the master secret key $\text{MSK}_{\text{ABS}} = (K, \text{HK})$ and a signing policy TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle \in \mathbb{M}_\lambda$, the setup authority performs the following steps:

Constants: PPRF key K
Inputs: SSB hash value h, Length ℓ_{INP}
Output: SIG verification key $\widehat{\text{VK}}_{\text{SIG}}$
1. Compute $\hat{r}_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\widehat{\text{SK}}_{\text{SIG}}, \widehat{\text{VK}}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; \hat{r}_{\text{SIG}})$.
2. Output $\widehat{\text{VK}}_{\text{SIG}}$.

Fig. 3.1. $\mathsf{Verify.Prog}_{\text{ABS}}$

1. At first, it selects PPRF keys $K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it forms $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. Then, it constructs the obfuscated programs
   - $\mathcal{P}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0, w_0, v_0, K_{\text{SPS},E}])$,
   - $\mathcal{P}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_{\text{ABS}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}[M, T = 2^\lambda, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}])$,

   where the programs Init-SPS.Prog, Accumulate.Prog, Change-SPS.Prog and $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}$ are as depicted respectively in Figs. 3.2, 3.3, 3.4 and 3.5.
4. It provides the constrained key $\text{SK}_{\text{ABS}}(M) = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{ABS}})$ to a legitimate signer.

Constants: Initial TM state q_0, Accumulator value w_0, Iterator value v_0, PPRF key $K_{\text{SPS},E}$
Input: SSB hash value h
Output: Signature $\sigma_{\text{SPS,OUT}}$
1. Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, 0))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
2. Output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},E}, (v_0, q_0, w_0, 0))$.

Fig. 3.2. Init-SPS.Prog

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF key $K_{\text{SPS},E}$
Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}
Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$
1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. 3.3. Accumulate.Prog

Constants: PPRF keys $K_{\text{SPS},A}, K_{\text{SPS},E}$
Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$
Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$
1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Set $m = (v, \text{ST}, w, 0)$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. 3.4. Change-SPS.Prog

$\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg}) \to \sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}})$ or $\perp$: A signer takes as input its signing key $\text{SK}_{\text{ABS}}(M) = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{ABS}})$, corresponding to its legitimate signing policy TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle \in \mathbb{M}_\lambda$, an attribute string $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{U}_{\text{ABS}}$ with $|x| = \ell_x$, and a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$. If $M(x) = 0$, it outputs $\perp$. Otherwise, it proceeds as follows:
1. It first computes $h = \mathcal{H}_{\text{HK}}(x)$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}$

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: (SIG signing key SK_{SIG}, SIG verification key VK_{SIG}), or (Header Position POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SSB.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4. (a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6. (a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. 3.5. $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}$

2. Next, it computes $\breve{\sigma}_{\text{SPS},0} = \mathcal{P}_1(h)$.
3. Then for $j = 1, \ldots, \ell_x$, it iteratively performs the following:
   (a) It computes $\pi_{\text{SSB},j-1} \xleftarrow{\$} \mathsf{SSB.Open}(\text{HK}, x, j-1)$.
   (b) It computes $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$.
   (c) It computes $\text{OUT} = \mathcal{P}_2(j-1, x_{j-1}, q_0, w_{j-1}, \text{AUX}_j, v_{j-1}, \breve{\sigma}_{\text{SPS},j-1}, h, \pi_{\text{SSB},j-1})$.
   (d) If $\text{OUT} = \perp$, it outputs OUT. Else, it parses OUT as $\text{OUT} = (w_j, v_j, \breve{\sigma}_{\text{SPS},j})$.
   (e) It computes $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1, x_{j-1})$.
4. It computes $\sigma_{\text{SPS},0} = \mathcal{P}_3(q_0, w_{\ell_x}, v_{\ell_x}, h, \ell_x, \breve{\sigma}_{\text{SPS},\ell_x})$.
5. It sets $\text{POS}_{M,0} = 0$ and $\text{SEED}_0 = \epsilon$.
6. Suppose, M accepts x in t_x steps. For $t = 1, \ldots, t_x$, it iteratively performs the following steps:
   (a) It computes $(\text{SYM}_{M,t-1}, \pi_{\text{ACC},t-1}) = \mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (b) It computes $\text{AUX}_{\ell_x+t} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (c) It computes $\text{OUT} = \mathcal{P}_{\text{ABS}}(t, \text{SEED}_{t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t-1}, \text{ST}_{M,t-1}, w_{\ell_x+t-1}, \pi_{\text{ACC},t-1}, \text{AUX}_{\ell_x+t}, v_{\ell_x+t-1}, h, \ell_x, \sigma_{\text{SPS},t-1})$.
   (d) If $t = t_x$, it parses OUT as $\text{OUT} = (\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$. Otherwise, it parses OUT as $\text{OUT} = (\text{POS}_{M,t}, \text{SYM}_{M,t}^{(\text{WRITE})}, \text{ST}_{M,t}, w_{\ell_x+t}, v_{\ell_x+t}, \sigma_{\text{SPS},t}, \text{SEED}_t)$.
   (e) It computes $\text{STORE}_{\ell_x+t} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t}^{(\text{WRITE})})$.
7. Finally, it computes $\sigma_{\text{SIG}} \xleftarrow{\$} \mathsf{SIG.Sign}(\text{SK}_{\text{SIG}}, \mathsf{msg})$.
8. It outputs the signature $\sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}}) \in \mathcal{S}_{\text{ABS}}$.

$\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x, \mathsf{msg}, \sigma_{\text{ABS}}) \to \hat{\beta} \in \{0,1\}$: A verifier takes as input the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{V}_{\text{ABS}})$, an attribute string $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{U}_{\text{ABS}}$, where $|x| = \ell_x$, a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, together with a signature $\sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}}) \in \mathcal{S}_{\text{ABS}}$. It executes the following:

1. It first computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. Next, it computes $\widehat{\text{VK}}_{\text{SIG}} = \mathcal{V}_{\text{ABS}}(h, \ell_x)$.
3. If $[\text{VK}_{\text{SIG}} = \widehat{\text{VK}}_{\text{SIG}}] \wedge [\mathsf{SIG.Verify}(\text{VK}_{\text{SIG}}, \mathsf{msg}, \sigma_{\text{SIG}}) = 1]$, it outputs 1. Otherwise, it outputs 0.

3.4 Security Analysis

Theorem 3.1 (Security of the ABS Scheme of Section 3.3). *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$*, $\mathcal{F}$ is a secure puncturable pseudorandom function,* SSB *is a somewhere statistically binding hash function,* ACC *is a secure positional accumulator,* ITR *is a secure cryptographic iterator,* SPS *is a secure splittable signature scheme,* PRG *is a secure injective pseudorandom generator, and* SIG *is existentially unforgeable against chosen message attack, the* ABS *scheme of Section 3.3 satisfies signer privacy and existential unforgeability against selective attribute adaptive chosen message attack.*

■ **Proof Overview**

The *signer privacy* property of our ABS construction follows readily from the following observation: Note that the signatures only depend on the PPRF key K, the SSB hash of the signing attribute strings with respect to which the signatures are issued, and the signed messages. The latter two have no connection with the signing keys at all, while the PPRF key K is shared among all the signing keys.

The proof of *existential unforgeability* is rather complicated. This is where we employ our new technical ideas to enrich the techniques of [21, 10] to deal with adaptive key queries of the adversary. The actual proof of unforgeability involves many subtleties. However, here we would like to sketch a bird's eye-view of our proof ideas. In order to prove unforgeability in selective attribute adaptive chosen message attack model described in Section 3.1, we aim to modify the signing keys given to the adversary $\mathcal{A}$ during the experiment to embed the punctured PPRF key $K\{(h^*, \ell^*)\}$ punctured at (h^*, ℓ^*) instead of the full PPRF key K, which is part of the master ABS key sampled by the challenger $\mathcal{B}$. Here, h^* and ℓ^* respectively denotes the SSB hash value (under the hash key HK sampled by $\mathcal{B}$ while performing the setup) and length of the challenge signing attribute string x^* submitted by the adversary $\mathcal{A}$. Once this substitution is made, the unforgeability can be argued employing the selective pseudorandomness of the PPRF $\mathcal{F}$ and the existential unforgeability of the digital signature scheme SIG. Now, in order to make this substitution, it is to be ensured that the obfuscated next step programs included in the constrained keys never evaluates the PPRF $\mathcal{F}$ for inputs corresponding to (h^*, ℓ^*) even if reaching the accepting state. Our proof transforms the signing keys one at a time through multiple hybrid steps. Suppose that the total number of signing keys queried by $\mathcal{A}$ be $\hat{q}_{\text{KEY}}$. Consider the transformation of the ν^{th} signing key $(1 \leq \nu \leq \hat{q}_{\text{KEY}})$ corresponding to the TM $M^{(\nu)}$ that runs on the challenge signing attribute string x^* for $t^{*(\nu)}$ steps and reaches the rejecting state. In the course of transformation, the obfuscated next step program $\mathcal{P}^{(\nu)}_{\text{ABS}}$ of the ν^{th} signing key is first altered to one that never evaluates the PPRF $\mathcal{F}$ for inputs corresponding to (h^*, ℓ^*) within the first $t^{*(\nu)}$ steps. The idea of transforming the signing keys in this way is analogous to that of [21, 10].

However, [21, 10] could achieve the key transition only based on the properties of positional accumulators and splittable signatures. At a very high level, [21, 10] used a system-wide public parameters for the positional accumulators and used it as the tool for compressing the arbitrary length input strings. Unfortunately, the technique of [21,10] does not work if the key queries are adaptive as in our case. This is because, while performing the transition of the ν^{th} queried signing key, the challenger $\mathcal{B}$ at various stages needs to generate the accumulator public parameters in read/write enforcing mode where the enforcing property is tailored to the steps of execution of the specific TM $M^{(\nu)}$ on x^*. Evidently, $\mathcal{B}$ can determine those execution steps only after receiving the ν^{th} signing key query from $\mathcal{A}$. However, if a system-wide set of public parameters for the positional accumulator is used, then $\mathcal{B}$ would also require it while creating the signing keys queried by $\mathcal{A}$ before making the ν^{th} signing key query. Thus, it is immediate that $\mathcal{B}$ must generate the set of public parameters for positional accumulator *prior to receiving* the ν^{th} query from $\mathcal{A}$. This is *impossible* as setting the accumulator public parameters in read/write enforcing mode requires the knowledge of the TM $M^{(\nu)}$, which is *not available* before the ν^{th} signing key query of $\mathcal{A}$. We resolve this issue by generating distinct set of public parameters of the positional accumulator for each signing key. However, we must provision for a system-wide compressing tool for compressing the arbitrary-length signing attribute strings. Here, the SSB hash comes to our rescue. However, using two different kinds of compressing tools, one system-wide and the other signing key specific, causes additional complications in the security proof. We overcome those challenges by using several tricks, which would be clear while going through our formal unforgeability proof.

We follow the same novel technique introduced in [10] for handling the tail hybrids in the final stage of transformation of the signing keys in our unforgeability experiment. Note that as in [10], we are also considering TM's which run for at most $T = 2^{\lambda}$ steps on any input. Unlike [21], the authors of [10] have devised an elegant approach to obtain an end to end polynomial reduction to the security of IO for the tail hybrids by means of an injective pseudorandom generator (PRG). We directly adopt that technique to deal with the tail hybrids in our unforgeability proof. A high level overview of the approach is outlined below.

Let us call the time step 2^{τ} as the τ^{th} landmark and the interval $[2^{\tau}, 2^{\tau+1} - 1]$ as the τ^{th} interval. Like [10], our obfuscated next step programs $\mathcal{P}_{\text{ABS}}$ included within the signing keys take an additional PRG seed as input at each time step, and perform some additional checks on the input PRG seed. At time steps just before a landmark, the programs output a new pseudorandomly generated PRG seed, which is then used in the next interval. Using standard IO techniques, it can be shown that for inputs corresponding to (h^*, ℓ^*), if the program $\mathcal{P}_{\text{ABS}}$ outputs $\perp$, for all time steps upto the one just before a

landmark, then we can alter the program indistinguishably so that it outputs $\perp$ at all time steps in the next interval. Employing this technique, we can move across an exponential number of time steps at a single switch of the next step program $\mathcal{P}_{\text{ABS}}$.

■ Formal Proof

► **Signer Privacy**: Observe that for any message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, $(\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$, $\text{MSK}_{\text{ABS}} = (K, \text{HK})) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$, and $x \in \mathcal{U}_{\text{ABS}}$ with $|x| = \ell_x$, a signature on msg under x is of the form $\sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}})$, where $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; \mathcal{F}(K, (\mathcal{H}_{\text{HK}}(x), \ell_x)))$, $\sigma_{\text{SIG}} = \mathsf{SIG.Sign}(\text{SK}_{\text{SIG}}, \mathsf{msg})$. Here, $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$. Thus, the distribution of the signature σ_{ABS} is clearly the same regardless of the signing key $\text{SK}_{\text{ABS}}(M)$ that is used to compute it.

► **Existential Unforgeability**: We will prove the existential unforgeability of the ABS construction of Section 3.3 against selective attribute adaptive chosen message attack by means of a sequence of hybrid experiments. We will demonstrate based on the security of various primitives that the advantage of any PPT adversary $\mathcal{A}$ in consecutive hybrid experiments differs only negligibly as well as that in the final hybrid experiment is negligible. We note that due to the selective attribute setting, the challenger $\mathcal{B}$ knows the challenge attribute string $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{U}_{\text{ABS}}$ and the SSB hash value $h^* = \mathcal{H}_{\text{HK}}(x^*)$ before receiving any signing key or signature query from the adversary $\mathcal{A}$. Suppose, the total number of signing key query and signature query made by the adversary $\mathcal{A}$ be $\hat{q}_{\text{KEY}}$ and $\hat{q}_{\text{SIGN}}$ respectively. As noted in Remark 3.1, without loss of generality we will assume that $\mathcal{A}$ only queries signatures on messages under the challenge attribute string x^*. The description of the hybrid experiments follows:

Sequence of Hybrid Experiments

Hyb_0: This experiment corresponds to the real selective attribute adaptive chosen message unforgeability experiment described in Section 3.1. More precisely, this experiment proceeds as follows:

- $\mathcal{A}$ submits a challenge attribute string $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{U}_{\text{ABS}}$ with $|x^*| = \ell^*$ to $\mathcal{B}$.
- $\mathcal{B}$ generates $(\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K])), \text{MSK}_{\text{ABS}} = (\text{HK}, K)) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$, as described in Section 3.3, and provides PP_{ABS} to $\mathcal{A}$.
- For $\eta = 1, \dots, \hat{q}_{\text{KEY}}$, in response to the η^{th} signing key query corresponding to signing policy TM $M^{(\eta)} = \langle Q^{(\eta)}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta^{(\eta)}, q_0^{(\eta)}, q_{\text{AC}}^{(\eta)}, q_{\text{REJ}}^{(\eta)} \rangle \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ creates

$$\text{SK}_{\text{ABS}}(M^{(\eta)}) = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \dots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}]) \end{array} \right) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M^{(\eta)}),$$

  as described in Section 3.3 and returns $\text{SK}_{\text{ABS}}(M^{(\eta)})$ to $\mathcal{A}$.
- For $\theta = 1, \dots, \hat{q}_{\text{SIGN}}$, in reply to the θ^{th} signature query on message $\mathsf{msg}^{(\theta)}$ under attribute string x^*, $\mathcal{B}$ identifies some TM $M^* \in \mathbb{M}_\lambda$ such that $M^*(x^*) = 1$, generates $\text{SK}_{\text{ABS}}(M^*) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M^*)$, and computes $\sigma_{\text{ABS}}^{(\theta)} = (\text{VK}_{\text{SIG}}^*, \sigma_{\text{SIG}}^{(\theta)}) \xleftarrow{\$} \mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M^*), x^*, \mathsf{msg}^{(\theta)})$ as described in Section 3.3. $\mathcal{B}$ gives back $\sigma_{\text{ABS}}^{(\theta)}$ to $\mathcal{A}$.
- Finally, $\mathcal{A}$ outputs a forged signature σ_{ABS}^* on some message msg^* under attribute string x^*.

$\mathsf{Hyb}_{0,\nu}$ $(\nu = 1, \dots, \hat{q}_{\text{KEY}})$: This experiment is similar to Hyb_0 except that for $\eta \in [\hat{q}_{\text{KEY}}]$, in reply to the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ returns

the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]}) \end{pmatrix},$$

if $\underline{\eta \leq \nu}$, where the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}$ is an alteration of the program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}$ (Fig. 3.5) and is described in Fig. 3.6, while it returns the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}]) \end{pmatrix},$$

if $\underline{\eta > \nu}$. Observe that $\mathsf{Hyb}_{0,0}$ coincides with Hyb_0.

Constants: TM $M = \langle Q, \Sigma_{\mathrm{INP}}, \Sigma_{\mathrm{TAPE}}, \delta, q_0, q_{\mathrm{AC}}, q_{\mathrm{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator $\mathrm{PP}_{\mathrm{ACC}}$, Public parameters for iterator $\mathrm{PP}_{\mathrm{ITR}}$, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\mathrm{SPS},A}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String $\mathrm{SEED}_{\mathrm{IN}}$, Header position $\mathrm{POS}_{\mathrm{IN}}$, Symbol $\mathrm{SYM}_{\mathrm{IN}}$, TM state $\mathrm{ST}_{\mathrm{IN}}$, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\mathrm{SPS,IN}}$

Output: (SIG signing key $\mathrm{SK}_{\mathrm{SIG}}$, SIG verification key $\mathrm{VK}_{\mathrm{SIG}}$), or (Header Position $\mathrm{POS}_{\mathrm{OUT}}$, Symbol $\mathrm{SYM}_{\mathrm{OUT}}$, TM state $\mathrm{ST}_{\mathrm{OUT}}$, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\mathrm{SPS,OUT}}$, String $\mathrm{SEED}_{\mathrm{OUT}}$), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\mathrm{SEED}_{\mathrm{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\mathrm{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}, \pi_{\mathrm{ACC}}) = 0$, output $\perp$.
3. (a) Compute $r_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}, (h, \ell_{\mathrm{INP}}, t-1))$, $(\mathrm{SK}_{\mathrm{SPS},A}, \mathrm{VK}_{\mathrm{SPS},A}, \mathrm{VK}_{\text{SPS-REJ},A}) = \mathsf{SSB.Setup}(1^\lambda; r_{\mathrm{SPS},A})$.
   (b) Set $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}_{\mathrm{IN}}, w_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}})$. If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},A}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 0$, output $\perp$.
4. (a) Compute $(\mathrm{ST}_{\mathrm{OUT}}, \mathrm{SYM}_{\mathrm{OUT}}, \beta) = \delta(\mathrm{ST}_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}})$ and $\mathrm{POS}_{\mathrm{OUT}} = \mathrm{POS}_{\mathrm{IN}} + \beta$.
   (b) If $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{REJ}}$, output $\perp$.
   Else if $[\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}] \wedge [(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)]$, perform the following:
   (I) Compute $r_{\mathrm{SIG}} = \mathcal{F}(K, (h, \ell_{\mathrm{INP}}))$, $(\mathrm{SK}_{\mathrm{SIG}}, \mathrm{VK}_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\mathrm{SIG}})$.
   (II) Output $(\mathrm{SK}_{\mathrm{SIG}}, \mathrm{VK}_{\mathrm{SIG}})$.
   Else if $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}$, output $\perp$.
5. (a) Compute $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{OUT}}, \mathrm{POS}_{\mathrm{IN}}, \mathrm{AUX})$. If $w_{\mathrm{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}}, (\mathrm{ST}_{\mathrm{IN}}, w_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}))$.
6. (a) Compute $r'_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}, (h, \ell_{\mathrm{INP}}, t))$, $(\mathrm{SK}'_{\mathrm{SPS},A}, \mathrm{VK}'_{\mathrm{SPS},A}, \mathrm{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},A})$.
   (b) Set $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}_{\mathrm{OUT}}, w_{\mathrm{OUT}}, \mathrm{POS}_{\mathrm{OUT}})$. Compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},A}, m_{\mathrm{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\mathrm{SEED}_{\mathrm{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\mathrm{INP}}))$.
   Else, set $\mathrm{SEED}_{\mathrm{OUT}} = \epsilon$
8. Output $(\mathrm{POS}_{\mathrm{OUT}}, \mathrm{SYM}_{\mathrm{OUT}}, \mathrm{ST}_{\mathrm{OUT}}, w_{\mathrm{OUT}}, v_{\mathrm{OUT}}, \sigma_{\mathrm{SPS,OUT}}, \mathrm{SEED}_{\mathrm{OUT}})$.

Fig. 3.6. $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}$

Hyb_1: This experiment coincides with $\mathsf{Hyb}_{0,\hat{q}_{\mathrm{KEY}}}$. More formally, in this experiment for $\eta = 1, \ldots, \hat{q}_{\mathrm{KEY}}$, in reply to the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ generates all the components of the signing key as in Hyb_0, however, it returns the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{pmatrix}.$$

The rest of the experiment is analogous to Hyb_0.

Hyb_2: This experiment is identical to Hyb_1 other than the following exceptions:

(I) Upon receiving the challenge attribute string x^*, $\mathcal{B}$ proceeds as follows:
1. It selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ just as in Hyb_1,
2. It then computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$ and creates the punctured PPRF key $K\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, (h^*, \ell^*))$,
3. It computes $\widehat{r}^*_{\mathrm{SIG}} = \mathcal{F}(K, (h^*, \ell^*))$, forms $(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; \widehat{r}^*_{\mathrm{SIG}})$,
4. It sets the public parameters $\mathrm{PP}_{\mathrm{ABS}}$ to be given to $\mathcal{A}$ as $\mathrm{PP}_{\mathrm{ABS}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\mathrm{ABS}}[K\{(h^*, \ell^*)\}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}, h^*, \ell^*]))$, where the program $\mathsf{Verify.Prog}'_{\mathrm{ABS}}$ is an alteration of the program $\mathsf{Verify.Prog}_{\mathrm{ABS}}$ (Fig. 3.1) and is depicted in Fig. 3.7.

Constants: Punctured PPRF key $K\{(h^*, \ell^*)\}$, SIG verification key $\widehat{\mathrm{VK}}^*_{\mathrm{SIG}}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*
Inputs: SSB hash value h, Length ℓ_{INP}
Output: SIG verification key $\widehat{\mathrm{VK}}_{\mathrm{SIG}}$
(a) If $(h, \ell_{\mathrm{INP}}) = (h^*, \ell^*)$, output $\widehat{\mathrm{VK}}^*_{\mathrm{SIG}}$.
Else compute $\hat{r}_{\mathrm{SIG}} = \mathcal{F}(K\{(h^*, \ell^*)\}, (h, \ell_{\mathrm{INP}}))$, $(\widehat{\mathrm{SK}}_{\mathrm{SIG}}, \widehat{\mathrm{VK}}_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; \hat{r}_{\mathrm{SIG}})$.
(b) Output $\widehat{\mathrm{VK}}_{\mathrm{SIG}}$.

Fig. 3.7. $\mathsf{Verify.Prog}'_{\mathrm{ABS}}$

(II) For $\eta = 1, \ldots, \hat{q}_{\mathrm{KEY}}$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ provides $\mathcal{A}$ with the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, w^{(\eta)}_0, \mathrm{STORE}^{(\eta)}_0, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, v^{(\eta)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\eta)}_0, w^{(\eta)}_0, v^{(\eta)}_0, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\eta)}_{\mathrm{SPS},A}, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, K\{(h^*, \ell^*)\}, K^{(\eta)}_1, \ldots, K^{(\eta)}_\lambda, K^{(\eta)}_{\mathrm{SPS},A}, h^*, \ell^*]) \end{pmatrix}.$$

Hyb_3: This experiment is similar to Hyb_2 with the only exception that $\mathcal{B}$ selects $\hat{r}^*_{\mathrm{SIG}} \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$. More formally, this experiment has the following deviations from hyb_2:

(I) In this experiment $\mathcal{B}$ creates the punctured PPRF key $K\{(h^*, \ell^*)\}$ as in Hyb_2, however, it generates $(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}) \xleftarrow{\$} \mathsf{SIG.Setup}(1^\lambda)$. It includes the obfuscated program $\mathcal{IO}(\mathsf{Verify.Prog}'_{\mathrm{ABS}}[K\{(h^*, \ell^*)\}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}, h^*, \ell^*])$ within the public parameters $\mathrm{PP}_{\mathrm{ABS}}$ to be provided to $\mathcal{A}$ as earlier.
(II) Also, for $\theta = 1, \ldots, \hat{q}_{\mathrm{SIGN}}$, to answer the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma^{(\theta)}_{\mathrm{SIG}} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \mathsf{msg}^{(\theta)})$ and returns $\sigma^{(\theta)}_{\mathrm{ABS}} = (\widehat{\mathrm{VK}}^*_{\mathrm{SIG}}, \sigma^{(\theta)}_{\mathrm{SIG}})$ to $\mathcal{A}$.

Analysis

Let $\mathsf{Adv}^{(0)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)$ $(\nu = 1, \ldots, \hat{q}_{\mathrm{KEY}}), \mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda)$, and $\mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)$ represent respectively the advantage of the adversary $\mathcal{A}$, i.e., $\mathcal{A}$'s probability of successfully outputting a valid forgery, in $\mathsf{Hyb}_0, \mathsf{Hyb}_{0,\nu}$ $(\nu = 1, \ldots, \hat{q}_{\mathrm{KEY}}), \mathsf{Hyb}_1$, Hyb_2, and Hyb_3 respectively. Then, by the description of the hybrid experiments it follows that $\mathsf{Adv}^{\mathrm{ABS,UF\text{-}CMA}}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,0)}_{\mathcal{A}}(\lambda)$ and $\mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\hat{q}_{\mathrm{KEY}})}_{\mathcal{A}}(\lambda)$. Hence, we have

$$\mathsf{Adv}^{\mathrm{ABS,UF\text{-}CMA}}_{\mathcal{A}}(\lambda) \leq \sum_{\nu=1}^{\hat{q}_{\mathrm{KEY}}} |\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)| + \sum_{j=1}^{2} |\mathsf{Adv}^{(j)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(j+1)}_{\mathcal{A}}(\lambda)| + \mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda). \quad (3.1)$$

Lemmas A.1–A.4 presented in Appendix A will show that the RHS of Eq. (3.1) is negligible and thus the existential unforgeability of the ABS construction of Section 3.3 follows.

4 Conclusion

In this paper, we construct the *first* ABS scheme supporting signing policies expressible as *Turing machines* (TM) which can handle signing attribute strings of *arbitrary polynomial length*. On the technical side, we devise new ideas to empower the techniques of [21, 10] to deal with adaptive key queries.

References

1. Ananth, P., Jain, A., Lin, H., Matt, C., Sahai, A.: Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In: CRYPTO 2019. pp. 284–332. Springer
2. Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. Cryptology ePrint Archive, Report 2018/615
3. Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In: CRYPTO 2017. pp. 252–279. Springer
4. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im) possibility of obfuscating programs. In: CRYPTO 2001. pp. 1–18. Springer
5. Bellare, M., Fuchsbauer, G.: Policy-based signatures. In: PKC 2014. pp. 520–537. Springer
6. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: ASIACRYPT 2013. pp. 280–300. Springer
7. Boyle, E., Chung, K.M., Pass, R.: On extractability obfuscation. In: TCC 2014. pp. 52–73. Springer
8. Canetti, R., Chen, Y., Holmgren, J., Raykova, M.: Adaptive succinct garbled ram or: How to delegate your database. In: TCC 2016. pp. 61–90. Springer
9. Datta, P., Okamoto, T., Takashima, K.: Efficient attribute-based signatures for unbounded arithmetic branching programs. In: PKC 2019. pp. 127–158. Springer
10. Deshpande, A., Koppula, V., Waters, B.: Constrained pseudorandom functions for unconstrained inputs. In: EUROCRYPT 2016. pp. 124–153. Springer
11. Garg, S., Gentry, C., Halevi, S., Wichs, D.: On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In: CRYPTO 2014. pp. 518–535. Springer
12. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS 2013. pp. 40–49. IEEE
13. Gay, R., Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. Cryptology ePrint Archive, Report 2020/764
14. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM (JACM) 33(4), 792–807 (1986)
15. Hubacek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: ITCS 2015. pp. 163–172. ACM
16. Ishai, Y., Pandey, O., Sahai, A.: Public-coin differing-inputs obfuscation and its applications. In: TCC 2015. pp. 668–697. Springer
17. Jain, A., Lin, H., Matt, C., Sahai, A.: How to leverage hardness of constant-degree expanding polynomials over $\mathbb{R}$ to build $i\mathcal{O}$. In: EUROCRYPT 2019. pp. 251–281. Springer
18. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. Cryptology ePrint Archive, Report 2020/1003
19. Jain, A., Lin, H., Sahai, A.: Simplifying constructions and assumptions for $i\mathcal{O}$. Cryptology ePrint Archive, Report 2019/1252
20. Kaafarani, A.E., Katsumata, S.: Attribute-based signatures for unbounded circuits in the rom and efficient instantiations from lattices. Cryptology ePrint Archive, Report 2018/022
21. Koppula, V., Lewko, A.B., Waters, B.: Indistinguishability obfuscation for turing machines with unbounded memory. In: STOC 2015. pp. 419–428. ACM
22. Lin, H., Matt, C.: Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. Cryptology ePrint Archive, Report 2018/646
23. Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures. In: CT-RSA 2011. pp. 376–392. Springer
24. Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures: Achieving attribute-privacy and collusion-resistance. Cryptology ePrint Archive, Report 2008/328
25. Okamoto, T., Pietrzak, K., Waters, B., Wichs, D.: New realizations of somewhere statistically binding hashing and positional accumulators. In: ASIACRYPT 2015. pp. 121–145. Springer
26. Okamoto, T., Takashima, K.: Efficient attribute-based signatures for non-monotone predicates in the standard model. Cloud Computing, IEEE Transactions on 2(4), 409–421 (2014)
27. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC 2014. pp. 475–484. ACM

28. Sakai, Y., Attrapadung, N., Hanaoka, G.: Attribute-based signatures for circuits from bilinear map. In: PKC 2016. pp. 283–300. Springer
29. Sakai, Y., Katsumata, S., Attrapadung, N., Hanaoka, G.: Attribute-based signatures for unbounded languages from standard assumptions. In: ASIACRYPT 2018. pp. 493–522. Springer (2018)
30. Tang, F., Li, H., Liang, B.: Attribute-based signatures for circuits from multilinear maps. In: ISC 2014. pp. 54–71. Springer
31. Tsabary, R.: An equivalence between attribute-based signatures and homomorphic signatures, and new constructions for both. In: TCC 2018. pp. 489–518. Springer

Appendix

A Lemmas for the Proof of Theorem 3.1

Lemma A.1. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$*, $\mathcal{F}$ is a secure puncturable pseudorandom function,* SSB *is a somewhere statistically binding hash function,* ACC *is a secure positional accumulator,* ITR *is a secure cryptographic iterator,* SPS *is a secure splittable signature scheme, and* PRG *is a secure injective pseudorandom generator, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of our Lemma A.1 extends the ideas involved in the security proof for the message-hiding encoding of [21]. Lemma 1 in the security proof of the CPRF construction of [10] also employs a similar technique. However, as mentioned earlier, we elegantly extend the technique of [10] to support adaptive signing key queries of the adversary as stipulated in our unforgeability experiment. We will first provide a complete description of the sequence of hybrid experiments involved in the proof of Lemma A.1 and then provide the analysis of those hybrid experiments providing the details for only those segments which are technically distinct from [10].

Let $t^{*(\nu)}$ denotes the running time of the TM $M^{(\nu)} \in \mathbb{M}_\lambda$, queried by the adversary $\mathcal{A}$, on input the challenge string x^* and $2^{\tau^{*(\nu)}}$ be the smallest power of two greater than $t^{*(\nu)}$. The sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1}$ and $\mathsf{Hyb}_{0,\nu}$ are described below:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1}$ and $\mathsf{Hyb}_{0,\nu}$

$\mathsf{Hyb}_{0,\nu-1,0}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1}$.

$\mathsf{Hyb}_{0,\nu-1,1}$: This experiment if analogous to $\mathsf{Hyb}_{0,\nu-1,0}$ except that to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first picks PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \underline{K_{\text{SPS},B}^{(\nu)}}, K_{\text{SPS},E}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. After that, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. It provides $\mathcal{A}$ with the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]}) \end{array}\right),$$

where the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}$ is a modification of the program $\mathsf{Constrained\text{-}Key.Prog}'_{\text{ABS}}$ (Fig. 3.6) and is depicted in Fig. A.1.

$\mathsf{Hyb}_{0,\nu-1,2}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ performs the following steps:

1. It first chooses PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, \underline{K_{\text{SPS},F}^{(\nu)}} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. After that, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},B}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or (Header Position POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \wedge [(t > t^*) \vee (h \neq h^*) \vee (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4. (a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$B$'$]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6. (a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.1. $\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}$

3. It provides $\mathcal{A}$ with the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\overline{\mathsf{Accumulate.Prog}^{(1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]}), \\ \mathcal{IO}(\overline{\mathsf{Change\text{-}SPS.Prog}^{(1)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]}), \\ \mathcal{IO}(\overline{\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]}) \end{array} \right),$$

where the programs $\mathsf{Accumulate.Prog}^{(1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(1)}$ are modifications of the programs $\mathsf{Accumulate.Prog}$ and $\mathsf{Change\text{-}SPS.Prog}$ (Figs. 3.3 and 3.4) and are depicted in Figs. A.2 and A.3 respectively.

The rest of the experiment proceeds in the same way as in $\mathsf{Hyb}_{0,\nu-1,1}$.

$\mathsf{Hyb}_{0,\nu-1,3}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the PPRF keys as well as the public parameters for the positional accumulator and iterator just as in $\mathsf{Hyb}_{0,\nu-1,2}$, however, it returns the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\overline{\mathsf{Accumulate.Prog}^{(2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]}), \\ \mathcal{IO}(\overline{\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]}), \\ \mathcal{IO}(\overline{\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]}) \end{array} \right),$$

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(i \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.2. $\mathsf{Accumulate.Prog}^{(1)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m = (v, \text{ST}, w, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(\ell_{\text{INP}} \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{DSPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.3. $\mathsf{Change\text{-}SPS.Prog}^{(1)}$

where the programs $\mathsf{Accumulate.Prog}^{(2)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(2)}$ are modifications of the programs $\mathsf{Accumulate.Prog}^{(1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(1)}$ (Figs. A.2 and A.3) and are depicted in Figs. A.4 and A.5 respectively. The remaining part of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,2}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota}$ $(\iota = 0, \ldots, \ell^* - 1)$: In this hybrid experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3}$.
2. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1, \ldots, \iota$, it iteratively computes the following:
   - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
   - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
   - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
   - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

   It sets $m^{(\nu)}_{\iota,0} = (v^{(\nu)}_\iota, q^{(\nu)}_0, w^{(\nu)}_\iota, 0)$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(i > \ell^*) \vee (i = 0) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$.
   If $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.4. $\mathsf{Accumulate.Prog}^{(2)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m = (v, \text{ST}, w, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(\ell_{\text{INP}} > \ell^*) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{DSPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.5. $\mathsf{Change\text{-}SPS.Prog}^{(2)}$

3. It gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, \\ \qquad h^*, \ell^*]) \end{array} \right),$$

where the programs $\mathsf{Accumulate.Prog}^{(3,\iota)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}$ are alterations of the programs $\mathsf{Accumulate.Prog}^{(2)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(2)}$ (Figs. A.4 and A.5) and are described in Figs. A.6 and A.7 respectively.

The remaining part of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota'}$ $(\iota = 0, \ldots, \ell^* - 1)$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3}$ except that in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3}$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}, K_{\text{SPS},F}$, Message $m_{\iota,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (0 \leq i \leq \iota) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$.
   If $[(h, i) = (h^*, \iota)] \wedge [m_{\text{IN}} = m_{\iota,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
   Else if $[(h, i) = (h^*, \iota)] \wedge [m_{\text{IN}} \neq m_{\iota,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
   Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.6. $\mathsf{Accumulate.Prog}^{(3,\iota)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m = (v, \text{ST}, w, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(\ell_{\text{INP}} > \ell^*) \vee (0 < \ell_{\text{INP}} \leq \iota) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{DSPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.7. $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}$

2. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
   - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
   - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
   - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
   - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

   It sets $m^{(\nu)}_{\iota+1,0} = (v^{(\nu)}_{\iota+1}, q^{(\nu)}_0, w^{(\nu)}_{\iota+1}, 0)$.
3. It gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota')}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, m^{(\nu)}_{\iota+1,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \qquad h^*, \ell^*]) \end{array} \right),$$

where the program $\mathsf{Accumulate.Prog}^{(3,\iota')}$ is an alteration of the program $\mathsf{Accumulate.Prog}^{(3,\iota)}$ (Fig. A.6) and is shown in Fig. A.8.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}, K_{\text{SPS},F}$, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (0 \leq i \leq \iota) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$.
   If $[(h, i) = (h^*, \iota)] \wedge [m_{\text{OUT}} = m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
   Else if $[(h, i) = (h^*, \iota)] \wedge [m_{\text{OUT}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
   Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E} m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.8. $\mathsf{Accumulate.Prog}^{(3,\iota')}$

$\mathsf{Hyb}_{0,\nu-1,4}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,(\ell^*-1)'}$ with the exception that now in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ does not generate the PPRF key $K^{(\nu)}_{\text{SPS},F}$ and gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Accumulate.Prog}$ is shown in Fig. 3.3 while the program $\mathsf{Change\text{-}SPS.Prog}^{(4)}$, which is a modification of the program $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}$ (Fig. A.7), is depicted in Fig. A.9.

Constants: PPRF keys $K_{\text{SPS},A}, K_{\text{SPS},B}, K_{\text{SPS},E}$, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Set $m = (v, \text{ST}, w, 0)$.
   (c) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [m \neq m_{\ell^*,0}]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.9. $\mathsf{Change\text{-}SPS.Prog}^{(4)}$

$\mathsf{Hyb}_{0,\nu-1,4,\gamma}$ $(\gamma = 1, \ldots, t^{*(\nu)} - 1)$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4}$ except that in response

to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,4}$.
2. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \ell^*$, it iteratively computes the following:
   - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
   - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
   - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
   - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$
3. Then, $\mathcal{B}$ sets $\text{ST}_{M^{(\nu)},0} = q_0^{(\nu)}, \text{POS}_{M^{(\nu)},0} = 0$, and for $t = 1, \ldots, \gamma$, computes the following:
   - $(\text{SYM}_{M^{(\nu)},t-1}, \pi_{\text{ACC},t-1}^{(\nu)}) = \mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{\ell^*+t-1}^{(\nu)}, \text{POS}_{M^{(\nu)},t-1})$
   - $\text{AUX}_{\ell^*+t}^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{\ell^*+t-1}^{(\nu)}, \text{POS}_{M^{(\nu)},t-1})$
   - $(\text{ST}_{M^{(\nu)},t}, \text{SYM}_{M^{(\nu)},t}^{(\text{WRITE})}, \beta) = \delta^{(\nu)}(\text{ST}_{M^{(\nu)},t-1}, \text{SYM}_{M^{(\nu)},t-1})$
   - $w_{\ell^*+t}^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{\ell^*+t-1}^{(\nu)}, \text{SYM}_{M^{(\nu)},t}^{(\text{WRITE})}, \text{POS}_{M^{(\nu)},t-1}, \text{AUX}_{\ell^*+t}^{(\nu)})$
   - $v_{\ell^*+t}^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{\ell^*+t-1}^{(\nu)}, (\text{ST}_{M^{(\nu)},t-1}, w_{\ell^*+t-1}^{(\nu)}, \text{POS}_{M^{(\nu)},t-1}))$
   - $\text{STORE}_{\ell^*+t}^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{\ell^*+t-1}^{(\nu)}, \text{POS}_{M^{(\nu)},t-1}, \text{SYM}_{M^{(\nu)},t}^{(\text{WRITE})})$
   - $\text{POS}_{M^{(\nu)},t} = \text{POS}_{M^{(\nu)},t-1} + \beta$

   $\mathcal{B}$ sets $m_{\ell^*,\gamma}^{(\nu)} = (v_{\ell^*+\gamma}^{(\nu)}, \text{ST}_{M^{(\nu)},\gamma}, w_{\ell^*+\gamma}^{(\nu)}, \text{POS}_{M^{(\nu)},\gamma})$.
4. $\mathcal{B}$ provides $\mathcal{A}$ with the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(2,\gamma)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, \\ \quad m_{\ell^*,\gamma}^{(\nu)}, h^*, \ell^*]) \end{array} \right),$$

where the program Change-SPS.Prog is described in Fig. 3.4 and the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(2,\gamma)}$, a modification of program $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}$ (Fig. A.1), is described in Fig. A.10.

$\mathsf{Hyb}_{0,\nu-1,4,\gamma'}$ $(\gamma = 0, \ldots, t^{*(\nu)} - 1)$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,4}$ except that in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,4}$.
2. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \ell^*$, it iteratively computes the following:
   - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
   - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
   - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
   - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$
3. Then, $\mathcal{B}$ sets $\text{ST}_{M^{(\nu)},0} = q_0^{(\nu)}, \text{POS}_{M^{(\nu)},0} = 0$, and for $t = 1, \ldots, \gamma$, computes the following:
   - $(\text{SYM}_{M^{(\nu)},t-1}, \pi_{\text{ACC},t-1}^{(\nu)}) = \mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{\ell^*+t-1}^{(\nu)}, \text{POS}_{M^{(\nu)},t-1})$
   - $\text{AUX}_{\ell^*+t}^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{\ell^*+t-1}^{(\nu)}, \text{POS}_{M^{(\nu)},t-1})$
   - $(\text{ST}_{M^{(\nu)},t}, \text{SYM}_{M^{(\nu)},t}^{(\text{WRITE})}, \beta) = \delta^{(\nu)}(\text{ST}_{M^{(\nu)},t-1}, \text{SYM}_{M^{(\nu)},t-1})$
   - $w_{\ell^*+t}^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{\ell^*+t-1}^{(\nu)}, \text{SYM}_{M^{(\nu)},t}^{(\text{WRITE})}, \text{POS}_{M^{(\nu)},t-1}, \text{AUX}_{\ell^*+t}^{(\nu)})$

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},B}$, Message $m_{\ell^*,\gamma}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or (Header Position POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \wedge [(t > t^*) \vee (t \leq \gamma) \vee (h \neq h^*) \vee (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4. (a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$B$'$]$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$A$'$] \wedge [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [t \leq \gamma]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6. (a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   If $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, \gamma)] \wedge [m_{\text{OUT}} = m_{\ell^*,\gamma}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
   Else if $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, \gamma)] \wedge [m_{\text{OUT}} \neq m_{\ell^*,\gamma}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.10. $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma)}_{\text{ABS}}$

- $v^{(\nu)}_{\ell^*+t} = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{\ell^*+t-1}, (\text{ST}_{M^{(\nu)},t-1}, w^{(\nu)}_{\ell^*+t-1}, \text{POS}_{M^{(\nu)},t-1}))$
- $\text{STORE}^{(\nu)}_{\ell^*+t} = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{\ell^*+t-1}, \text{POS}_{M^{(\nu)},t-1}, \text{SYM}^{(\text{WRITE})}_{M^{(\nu)},t})$
- $\text{POS}_{M^{(\nu)},t} = \text{POS}_{M^{(\nu)},t-1} + \beta$

$\mathcal{B}$ sets $m^{(\nu)}_{\ell^*,\gamma} = (v^{(\nu)}_{\ell^*+\gamma}, \text{ST}_{M^{(\nu)},\gamma}, w^{(\nu)}_{\ell^*+\gamma}, \text{POS}_{M^{(\nu)},\gamma})$.

4. $\mathcal{B}$ provides $\mathcal{A}$ with the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma')}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad m^{(\nu)}_{\ell^*,\gamma}, h^*, \ell^*]) \end{array} \right),$$

where the program $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma')}_{\text{ABS}}$ is an alteration of the program $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma)}_{\text{ABS}}$ (Fig. A.10) and is described in Fig. A.11.

$\mathsf{Hyb}_{0,\nu-1,5}$: This experiment is similar to $\mathsf{Hyb}_{0,\nu-1,4,(t^{*(\nu)}-1)'}$ with the exception that in responding to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ gives $\mathcal{A}$ the signing

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},B}$, Message $m_{\ell^*,\gamma}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or (Header Position POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =\text{'-'}] \wedge [(t > t^*) \vee \underline{(t \leq \gamma + 1)} \vee (h \neq h^*) \vee (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4. (a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =\text{'}B\text{'}]$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =\text{'}A\text{'}] \wedge [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge \underline{[t \leq \gamma + 1]}$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6. (a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   If $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, \gamma + 1)] \wedge [m_{\text{IN}} = m_{\ell^*,\gamma}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
   Else if $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, \gamma + 1)] \wedge [m_{\text{IN}} \neq m_{\ell^*,\gamma}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.11. $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma')}_{\text{ABS}}$

key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(3)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad \underline{h^*, \ell^*}]) \end{array} \right),$$

where the program $\mathsf{Constrained\text{-}Key.Prog}^{(3)}_{\text{ABS}}$ is a modification of the program $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma')}_{\text{ABS}}$ (Fig. A.11) and is described in Fig. A.12.

$\mathsf{Hyb}_{0,\nu-1,6}$: This experiment corresponds to $\mathsf{Hyb}_{0,\nu}$.

Analysis

Let $\mathsf{Adv}^{(0,\nu-1,0)}_{\mathcal{A}}(\lambda)$, $\mathsf{Adv}^{(0,\nu-1,1)}_{\mathcal{A}}(\lambda)$, $\mathsf{Adv}^{(0,\nu-1,2)}_{\mathcal{A}}(\lambda)$, $\mathsf{Adv}^{(0,\nu-1,3)}_{\mathcal{A}}(\lambda)$, $\mathsf{Adv}^{(0,\nu-1,3,\iota)}_{\mathcal{A}}(\lambda)(\iota = 0, \ldots, \ell^* - 1)$, $\mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda)(\iota = 0, \ldots, \ell^* - 1)$, $\mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda)$, $\mathsf{Adv}^{(0,\nu-1,4,\gamma)}_{\mathcal{A}}(\lambda)(\gamma = 1, \ldots, t^{*(\nu)} - 1)$, $\mathsf{Adv}^{(0,\nu-1,4,\gamma')}_{\mathcal{A}}(\lambda)$ $(\gamma = 0, \ldots, t^{*(\nu)} - 1)$, $\mathsf{Adv}^{(0,\nu-1,5)}_{\mathcal{A}}(\lambda)$, and $\mathsf{Adv}^{(0,\nu-1,6)}_{\mathcal{A}}(\lambda)$ represent respectively the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in the hybrid experiment Hyb_Υ with Υ as indicated in the superscript of the advantage notation. By the description of the hybrid experiments it follows that $\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) \equiv$

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},B}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or (Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4. (a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [t \leq t^*]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6. (a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   If $(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, t^*)$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.12. $\mathsf{Constrained\text{-}Key.Prog}^{(3)}_{\text{ABS}}$

$\mathsf{Adv}^{(0,\nu-1,0)}_{\mathcal{A}}(\lambda)$ and $\mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,6)}_{\mathcal{A}}(\lambda)$. Thus we have,

$$
\begin{aligned}
&|\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)| \leq \\
&|\mathsf{Adv}^{(0,\nu-1,0)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,1)}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(0,\nu-1,1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,2)}_{\mathcal{A}}(\lambda)| + \\
&|\mathsf{Adv}^{(0,\nu-1,2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3)}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(0,\nu-1,3)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,0)}_{\mathcal{A}}(\lambda)| + \\
&\sum_{\iota=0}^{\ell^*-1} |\mathsf{Adv}^{(0,\nu-1,3,\iota)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda)| + \sum_{\iota=0}^{\ell^*-2} |\mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota+1)}_{\mathcal{A}}(\lambda)| + \\
&|\mathsf{Adv}^{(0,\nu-1,3,(\ell^*-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,0')}_{\mathcal{A}}(\lambda)| + \\
&\sum_{\gamma=1}^{t^{*(\nu)}-1} |\mathsf{Adv}^{(0,\nu-1,4,(\gamma-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,\gamma)}_{\mathcal{A}}(\lambda)| + \sum_{\gamma=1}^{t^{*(\nu)}-1} |\mathsf{Adv}^{(0,\nu-1,4,\gamma)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,\gamma')}_{\mathcal{A}}(\lambda)| + \\
&|\mathsf{Adv}^{(0,\nu-1,4,(t^{*(\nu)}-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,5)}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(0,\nu-1,5)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,6)}_{\mathcal{A}}(\lambda)|.
\end{aligned}
\tag{A.1}
$$

Lemmas B.1–B.12 provided in Appendix B will prove that the RHS of Eq. (A.1) is negligible and hence Lemma A.1 follows. □

Lemma A.2. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$ *and $\mathcal{F}$ satisfies the correctness under puncturing property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The two differences between Hyb_1 and Hyb_2 are the following:

(I) In Hyb_1, $\mathcal{B}$ includes $\mathcal{IO}(V_0)$ within the public parameters PP_{ABS} provided to $\mathcal{A}$, whereas, in Hyb_2, $\mathcal{B}$ includes the program $\mathcal{IO}(V_1)$ within PP_{ABS}, where
- $(V_0) = \mathsf{Verify.Prog}_{\text{ABS}}[K]$ (Fig. 3.1),
- $(V_1) = \mathsf{Verify.Prog}'_{\text{ABS}}[K\{(h^*, \ell^*)\}, \widehat{\text{VK}}^*_{\text{ABS}}, h^*, \ell^*]$ (Fig. 3.7).

(II) For $\eta = 1, \ldots, \hat{q}_{\text{KEY}}$, the signing key $\text{SK}_{\text{ABS}}(M^{(\eta)})$ returned by $\mathcal{B}$ to $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, includes the program $\mathcal{IO}(P^{(\eta)}_0)$ in the experiment Hyb_1, while $\text{SK}_{\text{ABS}}(M^{(\eta)})$ includes the program $\mathcal{IO}(P^{(\eta)}_1)$ in Hyb_2, where
- $P^{(\eta)}_0 = \mathsf{Constrained\text{-}Key.Prog}'_{\text{ABS}}[M^{(\eta)}, T = 2^\lambda, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K, K^{(\eta)}_1, \ldots, K^{(\eta)}_\lambda, K^{(\eta)}_{\text{SPS},A}, h^*, \ell^*]$,

– $P_1^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]$, the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}$ being described in Fig. 3.6.

Now, observe that on input $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$, both the programs V_0 and V_1 operates in the same manner only that the latter one uses the punctured PPRF key $K\{(h^*, \ell^*)\}$ for computing the string $\hat{r}_{\mathrm{SIG}}$ instead of the full PPRF key K used by the former program. Therefore, by the correctness under puncturing property of PPRF $\mathcal{F}$, it follows that for all inputs $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$, both the programs have identical output. Moreover, on input (h^*, ℓ^*), V_1 outputs the hardwired SIG verification key $\widehat{\mathrm{VK}}_{\mathrm{SIG}}^*$ which is computed as $(\widehat{\mathrm{SK}}_{\mathrm{SIG}}^*, \widehat{\mathrm{VK}}_{\mathrm{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda; \hat{r}_{\mathrm{SIG}}^*)$, where $\hat{r}_{\mathrm{SIG}}^* = \mathcal{F}(K, (h^*, \ell^*))$. Notice that these values are exactly the same as those outputted V_0 on input (h^*, ℓ^*). Thus, the two programs are functionally equivalent.

Further, note that the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}$ computes $\mathcal{F}(K, (h, \ell_{\mathrm{INP}}))$ if and only if $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$. Thus, again by the correctness under puncturing property of PPRF $\mathcal{F}$, the programs $P_0^{(\eta)}$ and $P_1^{(\eta)}$ are functionally equivalent as well for all $\eta \in [\hat{q}_{\mathrm{KEY}}]$.

Thus the security of $\mathcal{IO}$, Lemma A.2 follows. Observe that to prove this lemma we would actually have to proceed through a sequence of intermediate hybrid experiments where in each hybrid experiment we switch the programs one at a time. □

Lemma A.3. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(3)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(3)}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge attribute string $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{U}_{\mathrm{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- After receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. $\mathcal{B}$ sends (h^*, ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, \ell^*)\}$ along with a challenge value $r^* \in \mathcal{Y}_{\mathrm{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$. $\mathcal{B}$ *implicitly* views the key K^* as the key K.
  3. Then $\mathcal{B}$ creates $(\widehat{\mathrm{SK}}_{\mathrm{SIG}}^*, \widehat{\mathrm{VK}}_{\mathrm{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda; r^*)$.
  4. Next, $\mathcal{B}$ sets the public parameters $\mathrm{PP}_{\mathrm{ABS}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\mathrm{ABS}}[K^*\{(h^*, \ell^*)\}, \widehat{\mathrm{VK}}_{\mathrm{SIG}}^*, h^*, \ell^*]))$ and gives it to $\mathcal{A}$.
- For $\eta = 1, \ldots, \hat{q}_{\mathrm{KEY}}$, to answer the η $^{\mathrm{th}}$ signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:
  1. At first, $\mathcal{B}$ chooses PPRF keys $K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, $\mathcal{B}$ generates $(\mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ returns $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K^*\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, \\ \quad h^*, \ell^*]) \end{pmatrix}.$$

- For $\theta = 1, \ldots, \hat{q}_{\mathrm{SIGN}}$, in response to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\mathrm{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}_{\mathrm{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\mathrm{ABS}}^{(\theta)} = (\widehat{\mathrm{VK}}_{\mathrm{SIG}}^*, \sigma_{\mathrm{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its PPRF selective pseudorandomness experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\mathrm{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its PPRF selective pseudorandomness experiment.

Notice that if $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$, then $\mathcal{B}$ perfectly simulates hyb_2. On the other hand, if $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, then $\mathcal{B}$ perfectly simulates Hyb_3. This completes the proof of Lemma A.3. □

Lemma A.4. *Assuming* SIG *is existentially unforgeable against* CMA*, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $\mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. Suppose that there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the existential unforgeability of SIG using $\mathcal{A}$ as a sub-routine. The description $\mathcal{B}$ is as follows:

- $\mathcal{B}$ receives a SIG verification key VK^*_{SIG} from its SIG existential unforgeability challenger $\mathcal{C}$. Then, $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ and receives a challenge attribute string $x^* = x^*_0 \dots x^*_{\ell^*-1} \in \mathcal{U}_{\text{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- After receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Next, it selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and creates the punctured PPRF key $K\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, (h^*, \ell^*))$.
  3. Next, $\mathcal{B}$ sets the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\text{ABS}}[K\{(h^*, \ell^*)\}, \text{VK}^*_{\text{SIG}}, h^*, \ell^*]))$ and gives it to $\mathcal{A}$.
- For $\eta = 1, \dots, \hat{q}_{\text{KEY}}$, to answer the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:
  1. At first, $\mathcal{B}$ chooses PPRF keys $K^{(\eta)}_1, \dots, K^{(\eta)}_\lambda, K^{(\eta)}_{\text{SPS},A}, K^{(\eta)}_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, $\mathcal{B}$ generates $(\text{PP}^{(\eta)}_{\text{ACC}}, w^{(\eta)}_0, \text{STORE}^{(\eta)}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}^{(\eta)}_{\text{ITR}}, v^{(\eta)}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ returns $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}(M^{(\eta)}) = \begin{pmatrix} \text{HK}, \text{PP}^{(\eta)}_{\text{ACC}}, w^{(\eta)}_0, \text{STORE}^{(\eta)}_0, \text{PP}^{(\eta)}_{\text{ITR}}, v^{(\eta)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\eta)}_0, w^{(\eta)}_0, v^{(\eta)}_0, K^{(\eta)}_{\text{SPS},E}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K^{(\eta)}_{\text{SPS},E}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\eta)}_{\text{SPS},A}, K^{(\eta)}_{\text{SPS},E}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{ABS}}[M^{(\eta)}, T = 2^\lambda, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K\{(h^*, \ell^*)\}, K^{(\eta)}_1, \dots, K^{(\eta)}_\lambda, K^{(\eta)}_{\text{SPS},A}, \\ \qquad h^*, \ell^*]) \end{pmatrix}.$$

- For $\theta = 1, \dots, \hat{q}_{\text{SIGN}}$, in response to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\text{ABS}}$ under attribute string x^*, $\mathcal{B}$ forwards the message $\mathsf{msg}^{(\theta)}$ to $\mathcal{C}$ and receives back a signature $\sigma^{(\theta)}_{\text{SIG}}$ on $\mathsf{msg}^{(\theta)}$ from $\mathcal{C}$. $\mathcal{B}$ provides, $\sigma^{(\theta)}_{\text{ABS}} = (\text{VK}^*_{\text{SIG}}, \sigma^{(\theta)}_{\text{SIG}})$ to $\mathcal{A}$.
- At the end of interaction, $\mathcal{A}$ outputs a signature $\sigma^*_{\text{ABS}} = (\widehat{\text{VK}}^*_{\text{SIG}}, \sigma^*_{\text{SIG}})$ on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $(\mathsf{msg}^*, \sigma^*_{\text{SIG}})$ as a forgery in its existential unforgeability experiment against SIG.

Observe that the simulation of the experiment Hyb_3 by $\mathcal{B}$ is perfect. Now, if $\mathcal{A}$ wins in the above simulated experiment, then the following must hold simultaneously:

(I) $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\text{ABS}}) = 1$.
(II) $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\text{SIGN}}]$.

Note that $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\text{ABS}}) = 1$ implies $[\widehat{\text{VK}}^*_{\text{SIG}} = \text{VK}^*_{\text{SIG}}] \wedge [\mathsf{SIG.Verify}(\widehat{\text{VK}}^*_{\text{SIG}}, \mathsf{msg}^*, \sigma^*_{\text{SIG}}) = 1]$, i.e., $\mathsf{SIG.Verify}(\text{VK}^*_{\text{SIG}}, \mathsf{msg}^*, \sigma^*_{\text{SIG}}) = 1$. Further, notice that $\mathsf{msg}^{(\theta)}$, for $\theta \in [\hat{q}_{\text{SIGN}}]$, are the only messages that $\mathcal{B}$ queried a signature on to $\mathcal{C}$. Thus, $(\mathsf{msg}^*, \sigma^*_{\text{SIG}})$ is indeed a valid forgery in the existential unforgeability experiment against SIG. □

B Lemmas for the proof of Lemma A.1

Lemma B.1. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* P/poly*,* $\mathcal{F}$ *is a secure puncturable pseudorandom function, and* SPS *is a splittable signature scheme satisfying* '$\text{VK}_{\text{SPS-REJ}}$ *indistinguishability', for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,0)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,1)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. To establish Lemma B.1, we introduce $t^{*(\nu)}+1$ intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,0}$ and $\mathsf{Hyb}_{0,\nu-1,1}$, namely, $\mathsf{Hyb}_{0,\nu-1,0,\gamma}$, for $\gamma \in [0, t^{*(\nu)}]$ such that $\mathsf{Hyb}_{0,\nu-1,0,t^{*(\nu)}}$ coincides with $\mathsf{Hyb}_{0,\nu-1,0}$ and $\mathsf{Hyb}_{0,\nu-1,0,0}$ coincides with $\mathsf{hyb}_{0,\nu-1,1}$.

Sequence of Intermediate Hybrids Between $\mathbf{Hyb_{0,\nu-1,0}}$ and $\mathbf{Hyb_{0,\nu-1,1}}$

$\mathbf{Hyb_{0,\nu-1,0,\gamma}}$ ($\boldsymbol{\gamma = 0, \ldots, t^{*(\nu)}}$): In this experiment in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first picks PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \underline{K_{\text{SPS},B}^{(\nu)}}, K_{\text{SPS},E}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. After that, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. It provides $\mathcal{A}$ with the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(0,\gamma)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, \\ \quad \underline{h^*, \ell^*}]) \end{array} \right),$$

where the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(0,\gamma)}$, depicted in Fig. B.1, is a modification of the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}$, shown in Fig. A.1.

The rest of the experiment is identical to $\mathsf{Hyb}_{0,\nu-1,0}$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}$, $K_{\text{SPS},B}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or (Header Position POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \wedge [(t > t^*) \vee \underline{(t \leq \gamma)} \vee (h \neq h^*) \vee (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4. (a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$B$'$]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6. (a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. B.1. $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(0,\gamma)}$

Analysis

Let us denote by $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,\gamma)}(\lambda)$ the advantage of $\mathcal{A}$, i.e., the absolute difference between 1/2 and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in the hybrid experiment $\mathsf{Hyb}_{0,\nu-1,0,\gamma}$, for $\gamma \in [0, t^{*(\nu)}]$. Clearly, $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,t^{*(\nu)})}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,0)}(\lambda)$. Hence, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda)| \leq \sum_{\gamma=1}^{t^{*(\nu)}} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,\gamma)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,\gamma-1)}(\lambda)|. \tag{B.1}$$

Claim B.1 below justifies that the RHS of Eq. (B.1) is negligible and consequently Lemma B.1 follows.

Claim B.1. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function, and* SPS *is a splittable signature scheme satisfying '$\mathrm{VK}_{\text{SPS-REJ}}$ indistinguishability', for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,\gamma)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,\gamma-1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim B.1 is similar to that of Claim B.1 of [10]. □

□

Lemma B.2. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function, and* SPS *is a splittable signature scheme satisfying '$\mathrm{VK}_{\text{SPS-REJ}}$ indistinguishability', for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. To prove Lemma B.2, we consider the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,1}$ and $\mathsf{Hyb}_{0,\nu-1,2}$:

Sequence of Intermediate Hybrids between $\mathbf{Hyb}_{0,\nu-1,1}$ and $\mathbf{Hyb}_{0,\nu-1,2}$

$\mathbf{Hyb}_{0,\nu-1,1,0}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,1}$.

$\mathbf{Hyb}_{0,\nu-1,1,1}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ selects an additional PPRF key $\underline{K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)}$ along with all the other PPRF keys as well as the public parameters for positional accumulator and iterator as generated in $\mathsf{Hyb}_{0,\nu-1,1,0}$, providing $\mathcal{A}$ with the signing key

$$\mathrm{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\overline{\mathsf{Accumulate.Prog}^{(0,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\text{ACC}}^{(\nu)}, \mathrm{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]}), \\ \mathcal{IO}(\overline{\mathsf{Change\text{-}SPS.Prog}^{(0,1)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]}), \\ \mathcal{IO}(\overline{\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\text{ACC}}^{(\nu)}, \mathrm{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]}) \end{array}\right),$$

where the programs $\mathsf{Accumulate.Prog}^{(0,1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(0,1)}$ are the alterations of the programs $\mathsf{Accumulate.Prog}^{(1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(1)}$ (Figs. A.2 and A.3) and are depicted in Figs. B.2 and B.3 respectively. The rest of the experiment proceeds in the same way as in $\mathsf{Hyb}_{0,\nu-1,1,0}$.

$\mathbf{Hyb}_{0,\nu-1,1,2}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator just as in $\mathsf{Hyb}_{0,\nu-1,1,1}$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h,i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h,i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) If $(h,i) = (h^*, \ell^*)$, set $\text{VK} = \text{VK}_{\text{SPS-REJ},F}$.
   (d) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (e) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (f) If $[\alpha =$'-'$] \wedge [(i \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (g) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. B.2. $\mathsf{Accumulate.Prog}^{(0,1)}$

Constants: PPRF keys $K_{\text{SPS},A}, K_{\text{SPS},B}, K_{\text{SPS},E}, K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) If $(h, \ell_{\text{INP}}) = (h^*, \ell^*)$, set $\text{VK} = \text{VK}_{\text{SPS-REJ},F}$.
   (d) Set $m = (v, \text{ST}, w, 0)$ and $\alpha =$'-'.
   (e) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (f) If $[\alpha =$'-'$] \wedge [(\ell_{\text{INP}} \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}, m, \sigma_{\text{SPS,IN}}) = 0]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (g) If $\alpha =$'-', output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{DSPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. B.3. $\mathsf{Change\text{-}SPS.Prog}^{(0,1)}$

2. Next, it creates the punctured PPRF key $K^{(\nu)}_{\text{SPS},F}\{(h^*,\ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},F}, (h^*,\ell^*))$ as well as computes $r^{(\nu,\ell^*)}_{\text{SPS},H} = \mathcal{F}(K^{(\nu)}_{\text{SPS},F}, (h^*,\ell^*))$ and $(\text{SK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}) = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,\ell^*)}_{\text{SPS},H})$.
3. It hands $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}\{(h^*,\ell^*)\}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}\{(h^*,\ell^*)\}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad h^*, \ell^*]) \end{array} \right),$$

where the programs $\mathsf{Accumulate.Prog}^{(0,2)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(0,2)}$ are the modifications of the programs $\mathsf{Accumulate.Prog}^{(0,1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(0,1)}$ (Figs. B.2 and B.3) and are described in Figs. B.4 and B.5 respectively.

The remaining part of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,1,1}$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF key $K_{\text{SPS},E}$, Punctured PPRF key $K_{\text{SPS},F}\{(h^*, \ell^*)\}$, Verification key VK_H, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \ell^*)\}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(i \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_H, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_H, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. B.4. $\mathsf{Accumulate.Prog}^{(0,2)}$

Constants: PPRF keys $K_{\text{SPS},A}, K_{\text{SPS},B}, K_{\text{SPS},E}$, Punctured PPRF key $K_{\text{SPS},F}\{(h^*, \ell^*)\}$, Verification key VK_H, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \ell^*)\}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(\ell_{\text{INP}} \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_H, m, \sigma_{\text{SPS,IN}}) = 0]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_H, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{DSPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. B.5. $\mathsf{Change\text{-}SPS.Prog}^{(0,2)}$

$\mathsf{Hyb}_{0,\nu-1,1,3}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,1,2}$ except that while creating the ν^{th} signing key queried by $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ selects $r^{(\nu,\ell^*)}_{\text{SPS},H} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, i.e., in other words, $\mathcal{B}$ generates $(\text{SK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, and gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \begin{pmatrix} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \ell^*)\}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \ell^*)\}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]) \end{pmatrix}.$$

$\mathsf{Hyb}_{0,\nu-1,1,4}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,1,3}$, however, it provides $\mathcal{A}$ with the

signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\ell^*)\}, \underline{\mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\ell^*)\}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, \overline{K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{pmatrix}.$$

The rest of the experiment is the same as $\mathsf{Hyb}_{0,\nu-1,1,3}$.

$\mathsf{Hyb}_{0,\nu-1,1,5}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ forms all the components as in $\mathsf{Hyb}_{0,\nu-1,1,4}$ except that it computes $\underline{r_{\mathrm{SPS},H}^{(\nu,\ell^*)} = \mathcal{F}(K_{\mathrm{SPS},F}^{(\nu)}, (h^*,\ell^*)), (\mathrm{SK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, \mathrm{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\mathrm{SPS},H}^{(\nu,\ell^*)})}$, and hands $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\ell^*)\}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\ell^*)\}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{pmatrix}.$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,1,4}$.

$\mathsf{Hyb}_{0,\nu-1,1,6}$: This experiment corresponds to $\mathsf{Hyb}_{0,\nu-1,2}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,\vartheta)}(\lambda)$ represents the advantage of $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,1,\vartheta}$, for $\vartheta \in [0,6]$. By definition, $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,0)}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,6)}(\lambda)$. Then, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda)| \leq \sum_{\vartheta=1}^{6} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,\vartheta-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,\vartheta)}(\lambda)|. \tag{B.2}$$

Claims B.2–B.7 below will demonstrate that the RHS of Eq. (B.2) is negligible and thus Lemma B.2 follows.

Claim B.2. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,1,0}$ and $\mathsf{Hyb}_{0,\nu-1,1,1}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,1,0}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P_0')$ within the ν^{th} signing key returned to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,1,1}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(P_1')$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]$ (Fig. 3.3),
- $P_0' = \mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]$ (Fig. 3.4),
- $P_1 = \mathsf{Accumulate.Prog}^{(0,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK},\ \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]$ (Fig. B.2),
- $P_1' = \mathsf{Change\text{-}SPS.Prog}^{(0,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]$ (Fig. B.3).

Now, observe that the programs P_0 and P_1 clearly have identical outputs for inputs corresponding to $(h,i) \neq (h^*,\ell^*)$. Also, by the correctness [Property (vii)] of splittable signature scheme SPS, both the programs output $\perp$ in case $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},E}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 0$ for inputs corresponding to (h^*,ℓ^*).

Thus the programs P_0 and P_1 are functionally equivalent. A similar argument justifies the functional equivalence of the programs P'_0 and P'_1.

Thus, by the security of $\mathcal{IO}$, Claim B.2 follows. Ofcourse, we need to consider a sequence of hybrid experiments to arrive at the result where in each hybrid experiment we change the programs one at a time. □

Claim B.3. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$ *and $\mathcal{F}$ satisfy the correctness under puncturing property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,1,1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,1,2)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,1,1}$ and $\mathsf{Hyb}_{0,\nu-1,1,2}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,1,1}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P'_0)$ within the ν^{th} signing key returned to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,1,2}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(P'_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(0,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]$ (Fig. B.2),
- $P'_0 = \mathsf{Change\text{-}SPS.Prog}^{(0,1)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]$ (Fig. B.3),
- $P_1 = \mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}\{(h^*,\ell^*)\}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}, h^*, \ell^*]$ (Fig. B.4),
- $P'_1 = \mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}\{(h^*,\ell^*)\}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}, h^*, \ell^*]$ (Fig. B.5).

Now, by the correctness under puncturing property of the PPRF $\mathcal{F}$, both the programs P_0 and P_1 have identical outputs on inputs corresponding to $(h,i) \neq (h^*,\ell^*)$. For inputs corresponding to (h^*,ℓ^*), P_1 uses the hardwired verification key $\text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}$, where in $\mathsf{Hyb}_{0,\nu-1,1,2}$, $\text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}$ is computed as $(\text{SK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}) = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,\ell^*)}_{\text{SPS},H})$ and $r^{(\nu,\ell^*)}_{\text{SPS},H} = \mathcal{F}(K^{(\nu)}_{\text{SPS},F}, (h^*,\ell^*))$. Observe that these values are exactly the same as those used by the program P_0 for inputs corresponding to (h^*,ℓ^*). Thus, both programs have identical outputs for inputs corresponding to (h^*,ℓ^*) as well. Hence, the two programs are functionally equivalent. A similar argument will justify that the programs P'_0 and P'_1 are functionally equivalent.

Therefore, by the security of $\mathcal{IO}$, Claim B.3 follows, considering a sequence of hybrid experiments where in each hybrid experiment we change the programs one at a time. □

Claim B.4. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,1,2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,1,3)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}^{(0,\nu-1,1,2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,1,3)}_{\mathcal{A}}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{U}_{\text{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r^*_{\text{SIG}} = \mathcal{F}(K, (h^*,\ell^*))$ followed by $(\widehat{\text{SK}}^*_{\text{SIG}}, \widehat{\text{VK}}^*_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda;\ r^*_{\text{SIG}})$.
  4. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,1,2}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ sends (h^*,ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*,\ell^*)\}$ and a value $r^* \in \mathcal{Y}_{\text{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*,\ell^*))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. $\mathcal{B}$ will *implicitly* view the key K^* as the key $K^{(\nu)}_{\text{SPS},F}$.
  4. $\mathcal{B}$ generates $(\text{SK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}) = \mathsf{SPS.Setup}(1^\lambda; r^*)$.

5. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K^*\{(h^*,\ell^*)\}, \mathrm{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}, \\ \qquad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K^*\{(h^*,\ell^*)\}, \mathrm{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \qquad h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\mathrm{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\mathrm{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}_{\mathrm{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\mathrm{ABS}}^{(\theta)} = (\widehat{\mathrm{VK}}_{\mathrm{SIG}}^*, \sigma_{\mathrm{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its PPRF selective pseudorandomness experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\mathrm{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its PPRF selective pseudorandomness experiment.

Note that if $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,1,2}$. On the other hand, if $r^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,1,3}$. This completes the proof of Claim B.4. □

Claim B.5. *Assuming* SPS *is a splittable signature scheme satisfying* '$\mathrm{VK}_{\text{SPS-REJ}}$ *indistinguishability', for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,4)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,4)}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the $\mathrm{VK}_{\text{SPS-REJ}}$ indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ receives a verification key VK of the splittable signature scheme SPS from its $\mathrm{VK}_{\text{SPS-REJ}}$ indistinguishability challenger $\mathcal{C}$, where VK is either a proper verification key $\mathrm{VK}_{\mathrm{SPS}}$ or a reject verification key $\mathrm{VK}_{\text{SPS-REJ}}$. Then, $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{U}_{\mathrm{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r_{\mathrm{SIG}}^* = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\mathrm{SK}}_{\mathrm{SIG}}^*, \widehat{\mathrm{VK}}_{\mathrm{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda; r_{\mathrm{SIG}}^*)$.
  4. $\mathcal{B}$ returns the public parameters $\mathrm{PP}_{\mathrm{ABS}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\mathrm{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\mathrm{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,1,3}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ creates the punctured PPRF key $K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\mathrm{SPS},F}, (h^*, \ell^*))$.
  4. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\ell^*)\}, \mathrm{VK}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\ell^*)\}, \mathrm{VK}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \qquad h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\text{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\text{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\text{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\text{SK}}_{\text{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\text{ABS}}^{(\theta)} = (\widehat{\text{VK}}_{\text{SIG}}^*, \sigma_{\text{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its $\text{VK}_{\text{SPS-REJ}}$ indistinguishability experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\text{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\text{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its $\text{VK}_{\text{SPS-REJ}}$ indistinguishability experiment.

Notice that if $\text{VK} = \text{VK}_{\text{SPS-REJ}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,1,3}$. On the other hand, if $\text{VK} = \text{VK}_{\text{SPS}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,1,4}$. This completes the proof of Claim B.5. □

Claim B.6. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,5)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim B.6 is similar to that of Claim B.4 with some appropriate changes which can be readily identified. □

Claim B.7. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$satisfies the correctness under puncturing property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,6)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim B.7 is analogous to that of Claim B.3 with some appropriate changes that are easy to determine. □

□

Lemma B.3. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function, and* SPS *is a splittable signature scheme satisfying '$\text{VK}_{\text{SPS-REJ}}$ indistinguishability', for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. To prove Lemma B.3, we consider the following sequence of ℓ^* intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,2}$ and $\mathsf{Hyb}_{0,\nu-1,3}$:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1,2}$ and $\mathsf{Hyb}_{0,\nu-1,3}$

$\mathsf{Hyb}_{0,\nu-1,2,\iota}$ ($\iota = 0, \ldots, \ell^* - 1$): In this experiment in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first chooses PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. After that, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. It provides $\mathcal{A}$ with the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\underline{\mathsf{Accumulate.Prog}^{(1,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]}), \\ \mathcal{IO}(\underline{\mathsf{Change\text{-}SPS.Prog}^{(1,\iota)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]}), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)} \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Accumulate.Prog}^{(1,\iota)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(1,\iota)}$ are the modifications of the programs $\mathsf{Accumulate.Prog}^{(2)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(2)}$ (Figs. A.4 and A.5) and are depicted in Figs. B.6 and B.7 respectively.

The rest of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,2}$. Observe that $\mathsf{Hyb}_{0,\nu-1,2,\ell^*-1}$ coincides with $\mathsf{hyb}_{0,\nu-1,2}$ and $\mathsf{Hyb}_{0,\nu-1,2,0}$ corresponds to $\mathsf{hyb}_{0,\nu-1,3}$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee \underline{(i \leq \iota)} \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. If $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. B.6. $\mathsf{Accumulate.Prog}^{(1,\iota)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(\ell_{\text{INP}} > \ell^*) \vee \underline{(\ell_{\text{INP}} \leq \iota)} \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. B.7. $\mathsf{Change\text{-}SPS.Prog}^{(1,\iota)}$

Analysis

Let us denote by $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota)}(\lambda)$ the advantage of $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in the hybrid experiment $\mathsf{Hyb}_{0,\nu-1,2,\iota}$, for $\iota \in [0, \ell^* - 1]$. Clearly, $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\ell^*-1)}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,0)}(\lambda)$. Hence we have,

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda)| \leq \sum_{\iota=1}^{\ell^*-1} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota-1)}(\lambda)|. \tag{B.3}$$

Claim B.8 below justifies that the RHS of Eq. (B.3) is negligible and consequently Lemma B.3 follows.

Claim B.8. *Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$, $\mathcal{F}$ is a secure puncturable pseudorandom function, and SPS is a splittable signature scheme satisfying '$\text{VK}_{\text{SPS-REJ}}$ indistinguishability', for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota-1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. The proof of Claim B.8 is similar to that of Lemma B.2 with some appropriate modifications which are easy to find out. □

□

Lemma B.4. *Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$, $\mathcal{F}$ is a secure puncturable pseudorandom function, and SPS is a splittable signature scheme satisfying '$\text{VK}_{\text{SPS-ONE}}$ indistinguishability', for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,0)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. In order to prove Lemma B.4, we consider the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,3}$ and $\mathsf{Hyb}_{0,\nu-1,3,0}$.

Sequence of Intermediate Hybrids between $\mathbf{Hyb}_{0,\nu-1,3}$ and $\mathbf{Hyb}_{0,\nu-1,3,0}$

$\mathbf{Hyb}_{0,\nu-1,3\text{-I}}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,3}$.

$\mathbf{Hyb}_{0,\nu-1,3\text{-II}}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys together with the public parameters for the positional accumulator and the iterator just as in $\mathsf{hyb}_{0,\nu-1,3\text{-I}}$.
2. After that, it creates the punctured PPRF key $K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},E},(h^*,0))$ as well as computes $r^{(\nu,0)}_{\text{SPS},G} = \mathcal{F}(K^{(\nu)}_{\text{SPS},E},(h^*,0))$ and $(\text{SK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},G} = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,0)}_{\text{SPS},G})$.
3. Then, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$ and computes $\sigma^{(\nu,0)}_{\text{SPS},G} = \mathsf{SPS.Sign}(\text{SK}^{(\nu,0)}_{\text{SPS},G}, m^{(\nu)}_{0,0})$.
4. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, \sigma^{(\nu,0)}_{\text{SPS},G}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Init\text{-}SPS.Prog}^{(1)}$ and $\mathsf{Accumulate.Prog}^{(2,1)}$ respectively are the alterations of the programs $\mathsf{Init\text{-}SPS.Prog}$ and $\mathsf{Accumulate.Prog}^{(2)}$ (Figs. 3.2 and A.4) and are depicted in Figs. B.8 and B.9.

The remaining part of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,3\text{-I}}$.

Constants: Initial TM state q_0, Accumulator value w_0, Iterator value v_0, Punctured PPRF key $K_{\text{SPS},E}\{(h^*,0)\}$, Signature σ_G, SSB hash value of challenge input h^*
Input: SSB hash value h
Output: Signature $\sigma_{\text{SPS,OUT}}$
1. If $h = h^*$, output σ_G.
   Else, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,0))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
2. Output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},E}, (v_0, q_0, w_0, 0))$.

Fig. B.8. $\mathsf{Init\text{-}SPS.Prog}^{(1)}$

$\mathbf{hyb}_{0,\nu-1,3\text{-III}}$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$ with the only exception that while constructing the ν^{th} signing key queried by $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ selects $r^{(\nu,0)}_{\text{SPS},G} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. More formally, to answer the ν^{th} signing key query of $\mathcal{A}$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$ except that it generates $(\text{SK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},G}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$, computes $\sigma^{(\nu,0)}_{\text{SPS},G} = \mathsf{SPS.Sign}(\text{SK}^{(\nu,0)}_{\text{SPS},G}, m^{(\nu)}_{0,0})$ and provides $\mathcal{A}$ with the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, \sigma^{(\nu,0)}_{\text{SPS},G}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]) \end{array}\right).$$

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF key $K_{\text{SPS},E}\{(h^*,0)\}$, PPRF key $K_{\text{SPS},F}$, Verification key VK_G, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) If $(h,i) \neq (h^*,0)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,i))$, $(\text{SK}_{\text{SPS},E},\text{VK}_{\text{SPS},E},\text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$. Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F},(h,i))$, $(\text{SK}_{\text{SPS},F},\text{VK}_{\text{SPS},F},\text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}},\text{ST},w_{\text{IN}},0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (i = 0) \vee (h \neq h^*)]$, output $\perp$. Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,i+1))$, $(\text{SK}'_{\text{SPS},E},\text{VK}'_{\text{SPS},E},\text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F},(h,i+1))$, $(\text{SK}'_{\text{SPS},F},\text{VK}'_{\text{SPS},F},\text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}},\text{ST},w_{\text{OUT}},0)$. If $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$. Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. B.9. $\mathsf{Accumulate.Prog}^{(2,1)}$

$\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$: This experiment is the same as $\mathsf{hyb}_{0,\nu-1,3\text{-III}}$ with the exception that in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds sa follows:

1. It first generates the full and punctured PPRF keys together with the public parameters for the positional accumulator and the iterator just as in $\mathsf{Hyb}_{0,\nu-1,3\text{-III}}$.
2. Next, it creates $(\text{SK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},G}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$, and forms $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{0,0},G}, \text{VK}^{(\nu,0)}_{\text{SPS-ONE},G}, \text{SK}^{(\nu,0)}_{\text{SPS-ABO},G}, \text{VK}^{(\nu,0)}_{\text{SPS-ABO},G}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,0)}_{\text{SPS},G}, m^{(\nu)}_{0,0})$.
3. $\mathcal{B}$ hands $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{0,0},G}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, \text{VK}^{(\nu,0)}_{\text{SPS-ONE},G}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad h^*, \ell^*]) \end{array}\right).$$

$\mathsf{Hyb}_{0,\nu-1,3\text{-V}}$: In this experiment, in reply to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components just as in $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$ and gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{0,0},G}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, \text{VK}^{(\nu,0)}_{\text{SPS-ONE},G}, m^{(\nu)}_{0,0}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Accumulate.Prog}^{(2,2)}$, described in Fig. B.10, is an alteration of the program $\mathsf{Accumulate.Prog}^{(2,1)}$ (Fig. B.9). The rest of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF key $K_{\text{SPS},E}\{(h^*,0)\}$, PPRF key $K_{\text{SPS},F}$, Verification key VK_G, Message $m_{0,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) If $(h,i) \neq (h^*,0)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,i))$, $(\text{SK}_{\text{SPS},E},\text{VK}_{\text{SPS},E},\text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$. Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
   (b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F},(h,i))$, $(\text{SK}_{\text{SPS},F},\text{VK}_{\text{SPS},F},\text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}},\text{ST},w_{\text{IN}},0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (i = 0) \vee (h \neq h^*)]$, output $\perp$. Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,i+1))$, $(\text{SK}'_{\text{SPS},E},\text{VK}'_{\text{SPS},E},\text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F},(h,i+1))$, $(\text{SK}'_{\text{SPS},F},\text{VK}'_{\text{SPS},F},\text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}},\text{ST},w_{\text{OUT}},0)$.
   If $[(h,i) = (h^*,0)] \wedge [m_{\text{IN}} = m_{0,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
   Else if $[(h,i) = (h^*,0)] \wedge [m_{\text{IN}} \neq m_{0,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
   Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E} m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. B.10. $\mathsf{Accumulate.Prog}^{(2,2)}$

$\mathsf{Hyb}_{0,\nu-1,3\text{-VI}}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the full and punctured PPRF keys as well as the public parameters for the positional accumulator and its iterator as in $\mathsf{Hyb}_{0,\nu-1,3\text{-V}}$.
2. Next, it forms $(\text{SK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},G}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$, and computes $\sigma^{(\nu,0)}_{\text{SPS},G} = \mathsf{SPS.Sign}(\text{SK}^{(\nu,0)}_{\text{SPS},G}, m^{(\nu)}_{0,0})$.
3. It provides $\mathcal{A}$ with the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, \sigma^{(\nu,0)}_{\text{SPS},G}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, m^{(\nu)}_{0,0}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad h^*, \ell^*]) \end{array}\right).$$

The remaining portion of the experiment is identical to $\mathsf{hyb}_{0,\nu-1,3\text{-V}}$.

$\mathsf{hyb}_{0,\nu-1,3\text{-VII}}$: In this experiment, while constructing the ν^{th} signing key queried by $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates everything just as in $\mathsf{Hyb}_{0,\nu-1,3\text{-VI}}$ except that it computes $r^{(\nu,0)}_{\text{SPS},G} = \mathcal{F}(K^{(\nu)}_{\text{SPS},E},(h^*,0))$, forms $(\text{SK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},G}) = \mathsf{SPS.Setup}(1^\lambda;\ r^{(\nu,0)}_{\text{SPS},G})$, and

provides $\mathcal{A}$ with the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, \sigma_{\mathrm{SPS},G}^{(\nu,0)}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,0)}, m_{0,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{pmatrix}.$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3\text{-VI}}$.

$\mathsf{Hyb}_{0,\nu-1,3\text{-VIII}}$: This experiment corresponds to $\mathsf{Hyb}_{0,\nu-1,3,0}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}\vartheta)}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,3\text{-}\vartheta}$, for $\vartheta \in \{\mathrm{I}, \ldots, \mathrm{VIII}\}$. Clearly, $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-I})}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,0)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-VIII})}(\lambda)$. Therefore, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,0)}(\lambda)| \leq \sum_{\vartheta=\mathrm{II}}^{\mathrm{VIII}} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}(\vartheta-\mathrm{I}))}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}\vartheta)}(\lambda)|. \tag{B.4}$$

Claims B.9–B.15 below will justify that the RHS of Eq. (B.4) is negligible and hence Lemma B.4 follows.

Claim B.9. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$ and $\mathcal{F}$ satisfies the correctness under puncturing property, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}I)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}II)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,3\text{-I}}$ and $\mathsf{hyb}_{0,\nu-1,3\text{-II}}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3\text{-I}}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P_0')$ within the ν^{th} signing key returned to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(p_1')$ instead, where

- $P_0 = \mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]$ (Fig. 3.2),
- $P_0' = \mathsf{Accumulate.Prog}^{(2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]$ (Fig. A.4),
- $P_1 = \mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, \sigma_{\mathrm{SPS},G}^{(\nu,0)}, h^*]$ (Fig. B.8),
- $P_1' = \mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,0)}, h^*, \ell^*]$ (Fig. B.9).

Now observe that the programs P_0 and P_1 are functionally equivalent since by the correctness under puncturing property of the PPRF $\mathcal{F}$, the PPRF output remains the same at all non-punctured points and at the point of puncturing, i.e., $(h^*, 0)$, the correct signature is hardwired in the program P_1. Similarly, P_0' and P_1' are also functionally equivalent by the correctness under puncturing property of $\mathcal{F}$ and the fact that at the point of puncturing i.e., $(h^*, 0)$ the correct verification key is hardwired into the program P_1'.

Therefore, by the security of $\mathcal{IO}$, Claim B.9 follows. □

Claim B.10. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}II)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}III)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-II})}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-III})}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{U}_{\mathrm{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.

- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r^*_{\mathrm{SIG}} = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r^*_{\mathrm{SIG}})$.
  4. $\mathcal{B}$ returns the public parameters $\mathrm{PP}_{\mathrm{ABS}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\mathrm{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\mathrm{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}, K^{(\nu)}_{\mathrm{SPS},B}, K^{(\nu)}_{\mathrm{SPS},F} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w^{(\nu)}_0, \mathrm{STORE}^{(\nu)}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ sends $(h^*, 0)$ as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, 0)\}$ and a value $r^* \in \mathcal{Y}_{\mathrm{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, 0))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$. $\mathcal{B}$ *implicitly* views the key K^* as the key $K^{(\nu)}_{\mathrm{SPS},E}$.
  4. $\mathcal{B}$ generates $(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},G}, \mathrm{VK}^{(\nu,0)}_{\mathrm{SPS},G}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-REJ},G}) = \mathsf{SPS.Setup}(1^\lambda; r^*)$.
  5. Then, $\mathcal{B}$ sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$ and computes $\sigma^{(\nu,0)}_{\mathrm{SPS},G} = \mathsf{SPS.Sign}(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},G}, m^{(\nu)}_{0,0})$.
  6. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w^{(\nu)}_0, \mathrm{STORE}^{(\nu)}_0, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^*\{(h^*,0)\}, \sigma^{(\nu,0)}_{\mathrm{SPS},G}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K^*\{(h^*,0)\}, K^{(\nu)}_{\mathrm{SPS},F}, \mathrm{VK}^{(\nu,0)}_{\mathrm{SPS},G}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\mathrm{SPS},A}, K^{(\nu)}_{\mathrm{SPS},B}, K^*\{(h^*,0)\}, K^{(\nu)}_{\mathrm{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\mathrm{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}, K^{(\nu)}_{\mathrm{SPS},B}, \\ \qquad h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\mathrm{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma^{(\theta)}_{\mathrm{SIG}} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma^{(\theta)}_{\mathrm{ABS}} = (\widehat{\mathrm{VK}}^*_{\mathrm{SIG}}, \sigma^{(\theta)}_{\mathrm{SIG}})$.
- Finally, $\mathcal{A}$ output a signature σ^*_{ABS} on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its PPRF selective pseudorandomness experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\mathrm{ABS}}) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its PPRF selective pseudorandomness experiment.

Note that if $r^* = \mathcal{F}(K^*, (h^*, 0))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$. On the other hand, if $r^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$, the $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3\text{-III}}$. This completes the proof of Claim B.10. □

Claim B.11. *Assuming* SPS *is a splittable signature scheme satisfying '*$\mathrm{VK}_{\text{SPS-ONE}}$*' indistinguishability, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,3\text{-}III)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3\text{-}IV)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}^{(0,\nu-1,3\text{-III})}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3\text{-IV})}_{\mathcal{A}}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the $\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{U}_{\mathrm{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r^*_{\mathrm{SIG}} = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r^*_{\mathrm{SIG}})$.
  4. $\mathcal{B}$ returns the public parameters $\mathrm{PP}_{\mathrm{ABS}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\mathrm{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\mathrm{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3\text{-III}}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}, K^{(\nu)}_{\mathrm{SPS},B}, K^{(\nu)}_{\mathrm{SPS},E}, K^{(\nu)}_{\mathrm{SPS},F} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.

2. Next, it generates $(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
3. Then, $\mathcal{B}$ creates the punctured PPRF key $K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\mathrm{SPS},E}^{(\nu)}, (h^*,0))$.
4. After that, $\mathcal{B}$ sends $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$ as the challenge message to its SPS $\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability challenger $\mathcal{C}$ and receives back a signature-verification key pair $(\sigma_{\text{SPS-ONE},m_{0,0}^{(\nu)}}, \mathrm{VK})$, where VK is either a normal verification key $\mathrm{VK}_{\mathrm{SPS}}$ or a one verification key $\mathrm{VK}_{\text{SPS-ONE}}$ for the message $m_{0,0}^{(\nu)}$.
5. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, \sigma_{\text{SPS-ONE},m_{0,0}^{(\nu)}}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \mathrm{VK}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \qquad h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\mathrm{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\mathrm{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}_{\mathrm{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\mathrm{ABS}}^{(\theta)} = (\widehat{\mathrm{VK}}_{\mathrm{SIG}}^*, \sigma_{\mathrm{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its $\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\mathrm{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its $\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability experiment.

Notice that if $\mathrm{VK} = \mathrm{VK}_{\mathrm{SPS}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3\text{-III}}$. On the other hand, if $\mathrm{VK} = \mathrm{VK}_{\text{SPS-ONE}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$. This completes the proof of Claim B.11. □

Claim B.12. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}IV)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}V)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$ and $\mathsf{hyb}_{0,\nu-1,3\text{-V}}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} signing key returned to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,3\text{-V}}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,0)}, h^*, \ell^*]$ (Fig. B.9),
- $P_1 = \mathsf{Accumulate.Prog}^{(2,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,0)}, m_{0,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.10).

Observe that the only inputs for which the programs P_0 and P_1 can possibly differ are those corresponding to $(h,i) = (h^*,0)$. However, the verification key hardwired in both the programs is $\mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,0)}$ which only accepts signature for $m_{\mathrm{IN}} = m_{0,0}^{(\nu)}$ by the correctness [Properties (i), (iii) and (v)]. This ensures that for inputs corresponding to $(h^*,0)$, if $m_{\mathrm{IN}} = m_{0,0}^{(\nu)}$ both the programs output an 'E' type signature, else, both output $\perp$. Thus, P_0 and P_1 are functionally equivalent.

Therefore, by the security of $\mathcal{IO}$, Claim B.12 follows. □

Claim B.13. *Assuming* SPS *is a splittable signature scheme satisfying '$\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability', for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}V)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VI)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim B.13 is similar to that of Claim B.11 with some readily identifiable modifications. □

Claim B.14. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VI)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim B.14 is analogous to that of Claim B.10 with some appropriate changes that are easy to find out. □

Claim B.15. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$ *and $\mathcal{F}$ satisfies the correctness under puncturing property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VII)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VIII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.15 is analogous to that of Claim B.9. □

□

Lemma B.5. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$, SSB *is a somewhere statistically binding hash function,* ACC *is a positional accumulator satisfying 'indistinguishability of write setup' and 'write enforcing' properties, as well as* ITR *is a secure cryptographic iterator, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota')}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. To prove Lemma B.5, we introduce the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,3,\iota}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$:

Sequence of Intermediate Hybrids between $\mathbf{Hyb_{0,\nu-1,3,\iota}}$ and $\mathbf{Hyb_{0,\nu-1,3,\iota'}}$

$\mathbf{Hyb_{0,\nu-1,3,\iota,0}}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,3,\iota}$.

$\mathbf{Hyb_{0,\nu-1,3,\iota,1}}$: In this experiment the challenger $\mathcal{B}$ forms the SSB hash key $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = \iota)$. The rest of the experiment proceeds in an analogous fashion to $\mathsf{Hyb}_{0,\nu-1,3,\iota,0}$.

$\mathbf{Hyb_{0,\nu-1,3,\iota,2}}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys and the public parameters for the iterator just as in $\mathsf{hyb}_{0,\nu-1,3,\iota,1}$.
2. Next, it forms $(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_\iota^*, \iota)))$.
3. After that, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \iota$, it iteratively computes the following:
   - $\mathrm{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1)$
   - $w_j^{(\nu)} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \mathrm{AUX}_j^{(\nu)})$
   - $\mathrm{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
   - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

   It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$.
4. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \qquad h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is similar to $\mathsf{hyb}_{0,\nu-1,3,\iota,1}$.

$\mathbf{hyb_{0,\nu-1,3,\iota,3}}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$.
2. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
   - $\mathrm{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1)$

– $w_j^{(\nu)} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \mathrm{AUX}_j^{(\nu)})$
– $\mathrm{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
– $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$ and $\underline{m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)}$.

3. It gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\underline{\mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]}), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Accumulate.Prog}^{(3,\iota,1)}$ is a modification of the program $\mathsf{Accumulate.Prog}^{(3,\iota)}$ (Fig. A.6) and is described in Fig. B.11.

The rest of the experiment if analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,4}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota,3}$ with the only exception that while construct-

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator $\mathrm{PP}_{\mathrm{ACC}}$, Public parameters for iterator $\mathrm{PP}_{\mathrm{ITR}}$, PPRF keys $K_{\mathrm{SPS},E}, K_{\mathrm{SPS},F}$, Messages $m_{\iota,0}$, $\underline{m_{\iota+1,0}}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol $\mathrm{SYM}_{\mathrm{IN}}$, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\mathrm{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) Compute $r_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}, (h,i))$, $(\mathrm{SK}_{\mathrm{SPS},E}, \mathrm{VK}_{\mathrm{SPS},E}, \mathrm{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},E})$.
   (b) Compute $r_{\mathrm{SPS},F} = \mathcal{F}(K_{\mathrm{SPS},F}, (h,i))$, $(\mathrm{SK}_{\mathrm{SPS},F}, \mathrm{VK}_{\mathrm{SPS},F}, \mathrm{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},F})$.
   (c) Set $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}, w_{\mathrm{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},E}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(i > \ell^*) \vee (0 \le i \le \iota) \vee (h \ne h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},F}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\mathrm{HK}, h, i, \mathrm{SYM}_{\mathrm{IN}}, \pi_{\mathrm{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, i, \mathrm{AUX})$. If $w_{\mathrm{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}}, (\mathrm{ST}, w_{\mathrm{IN}}, 0))$.
4. (a) Compute $r'_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}, (h, i+1))$, $(\mathrm{SK}'_{\mathrm{SPS},E}, \mathrm{VK}'_{\mathrm{SPS},E}, \mathrm{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},E})$.
   (b) Compute $r'_{\mathrm{SPS},F} = \mathcal{F}(K_{\mathrm{SPS},F}, (h, i+1))$, $(\mathrm{SK}'_{\mathrm{SPS},F}, \mathrm{VK}'_{\mathrm{SPS},F}, \mathrm{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},F})$.
   (c) Set $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}, w_{\mathrm{OUT}}, 0)$.
   If $[(h,i) = (h^*, \iota)] \wedge [(m_{\mathrm{IN}} = m_{\iota,0}) \wedge \underline{(m_{\mathrm{OUT}} = m_{\iota+1,0})}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},E}, m_{\mathrm{OUT}})$.
   Else if $[(h,i) = (h^*, \iota)] \wedge [\underline{(m_{\mathrm{IN}} \ne m_{\iota,0}) \vee (m_{\mathrm{OUT}} \ne m_{\iota+1,0})}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},F}, m_{\mathrm{OUT}})$.
   Else if $i < \ell^*$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},\alpha}, m_{\mathrm{OUT}})$.
   Else, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},E}, m_{\mathrm{OUT}})$.
5. Output $(w_{\mathrm{OUT}}, v_{\mathrm{OUT}}, \sigma_{\mathrm{SPS,OUT}})$.

Fig. B.11. $\mathsf{Accumulate.Prog}^{(3,\iota,1)}$

ing the ν^{th} signing key queried by $\mathcal{A}$, $\mathcal{B}$ generates $\underline{(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota,4}$ with the only exception that $\mathcal{B}$ generates $\underline{\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$.
2. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
   – $\mathrm{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1)$

- $w_j^{(\nu)} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \mathrm{AUX}_j^{(\nu)})$
- $\mathrm{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$

3. Then, it generates $\underline{(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda, ((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)))}$.
4. After that, for $j = 1, \ldots, \iota+1$, it iteratively computes $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$.
5. It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$ and $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.
6. It gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{pmatrix}.$$

The rest of the experiment if analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,7}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates everything as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$, however, it hands $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \underline{\mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota')}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*])}, \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{pmatrix},$$

where the program $\mathsf{Accumulate.Prog}^{(3,\iota')}$ is depicted in Fig. A.8. The rest of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,8}$: This experiment is analogous to $\mathsf{hyb}_{0,\nu-1,3,\iota,7}$ with the only exception that while constructing the ν^{th} signing key queried by $\mathcal{A}$, $\mathcal{B}$ generates $\underline{(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)}$. Notice that this experiment coincides with $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,\vartheta)}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,3,\iota,\vartheta}$, for $\vartheta \in [0, 8]$. From the description of the hybrid experiments it follows that $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,0)}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota')}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,8)}(\lambda)$. Hence, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota')}(\lambda)| \leq \sum_{\vartheta=1}^{8} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,\vartheta-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,\vartheta)}(\lambda)|. \tag{B.5}$$

Claims B.16–B.23 below will show that the RHS of Eq. (B.5) is negligible and thus Lemma B.5 follows.

Claim B.16. *Assuming* SSB *satisfies the 'index hiding' property, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,1)}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the index hiding property of SSB using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{U}_{\text{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ submits $n_{\text{SSB-BLK}} = 2^\lambda$ and the pair of indices $(i_0^* = 0, i_1^* = \iota)$ to its SSB index hiding challenger $\mathcal{C}$ and receives back a hash key HK, where either $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i_0^* = 0)$ or $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i_1^* = \iota)$.
  2. Next, $\mathcal{B}$ computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  3. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  4. After that, $\mathcal{B}$ computes $r_{\text{SIG}}^* = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\text{SK}}_{\text{SIG}}^*, \widehat{\text{VK}}_{\text{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}}^*)$.
  5. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,0}$.
- For $\theta \in [\hat{q}_{\text{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\text{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\text{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\text{SK}}_{\text{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\text{ABS}}^{(\theta)} = (\widehat{\text{VK}}_{\text{SIG}}^*, \sigma_{\text{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its SSB index hiding experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\text{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\text{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its SSB index experiment.

Note that if $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i_0^* = 0)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,0}$. On the other hand, if $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i_1^* = \iota)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,1}$. This completes the proof of Claim B.16. □

Claim B.17. *Assuming* ACC *is a positional accumulator satisfying the 'indistinguishability of write setup' property, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,2)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,2)}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the indistinguishability of write setup property of the positional accumulator ACC using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = \iota)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r_{\text{SIG}}^* = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\text{SK}}_{\text{SIG}}^*, \widehat{\text{VK}}_{\text{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda;\ r_{\text{SIG}}^*)$.
  4. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,1}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \dots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. After that, $\mathcal{B}$ sends $n_{\text{ACC-BLK}} = 2^\lambda$ and the sequence of symbol-index pairs $((x_0^*, 0), \dots, (x_\iota^*, \iota))$ to its ACC write setup indistinguishability challenger $\mathcal{C}$ and receives back $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$, where either $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ or $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enfoce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \dots, (x_\iota^*, \iota)))$.
  3. Next, it generates $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  4. Then, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0, 0)$. For $j = 1, \dots, \iota$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$
     - $w_j = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{j-1}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}\text{STORE}_{j-1}, j-1, x_{j-1}^*)$
     - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}, 0))$

     It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota, 0)$.

5. It gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\mathrm{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\mathrm{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}_{\mathrm{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\mathrm{ABS}}^{(\theta)} = (\widehat{\mathrm{VK}}_{\mathrm{SIG}}^*, \sigma_{\mathrm{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its ACC write setup indistinguishability experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\mathrm{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its ACC write setup indistinguishability experiment.

Note that if $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,1}$. On the other hand, if $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_\iota^*, \iota)))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$. This completes the proof of Claim B.17. □

Claim B.18. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, SSB possesses the 'somewhere statistically binding' property, and ACC is a positional accumulator having the 'write enforcing' property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,3)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota,3}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} signing key provided to $\mathcal{A}$, whereas, in $\mathsf{Hyb}_{0,\nu-1,3,\iota,3}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.6),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.11).

We will argue that the programs P_0 and P_1 are functionally equivalent, so that, by the security of $\mathcal{IO}$ Claim B.18 follows. The inputs on which the outputs of the two programs can possibly differ are those corresponding to $(h, i) = (h^*, \iota)$. For inputs corresponding to (h^*, ι), the program P_1 performs the additional check '$m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$' to determine the type of the outputted signature. We show that this check is redundant by demonstrating that for inputs corresponding to (h^*, ι), if $m_{\mathrm{IN}} = m_{\iota,0}^{(\nu)}$, then either both the programs output $\perp$ or it must hold that $m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$ and, therefore, both the programs output signatures of the same type. Notice that $m_{\mathrm{IN}} = m_{\iota,0}^{(\nu)}$ means $v_{\mathrm{IN}} = v_\iota^{(\nu)}, \mathrm{ST} = q_0^{(\nu)}$, and $w_{\mathrm{IN}} = w_\iota^{(\nu)}$. Thus, $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{\mathrm{IN}}, (\mathrm{ST}, w_{\mathrm{IN}}, 0)) = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_\iota^{(\nu)}, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)) = v_{\iota+1}^{(\nu)}$. Now, recall that in both experiments $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = \iota)$. Therefore, by the somewhere statistically binding property of SSB it follows that $\mathsf{SSB.Verify}(\mathrm{HK}, h^* = \mathcal{H}_{\mathrm{HK}}(x^*), \iota, \mathrm{SYM}_{\mathrm{IN}}, \pi_{\mathrm{SSB}}) = 1$ if and only if $\mathrm{SYM}_{\mathrm{IN}} = x_\iota^*$. Thus, for inputs corresponding to (h^*, ι), both programs will output $\perp$ in case $\mathrm{SYM}_{\mathrm{IN}} \neq x_\iota^*$. Further, in both the experiments, $(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_\iota^*, \iota)))$. Therefore, by the write enforcing property of ACC it follows that if $w_{\mathrm{IN}} = w_\iota^{(\nu)}$ and $\mathrm{SYM}_{\mathrm{IN}} = x_\iota^*$, then $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, \iota, \mathrm{AUX})$ results in $w_{\mathrm{OUT}} = w_{\iota+1}^{(\nu)}$ or $w_{\mathrm{OUT}} = \perp$. In case $w_{\mathrm{OUT}} = \perp$, then clearly both the programs output $\perp$. On the other hand, $w_{\mathrm{OUT}} = w_{\iota+1}^{(\nu)}$ implies $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}, w_{\mathrm{OUT}}, 0) = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0) = m_{\iota+1,0}^{(\nu)}$ and the two programs have identical outputs in this case as well. □

Claim B.19. *Assuming* ACC *is a positional accumulator satisfying the 'indistinguishability of write setup' property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,4)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim B.19 is similar to that of Claim B.17 with some appropriate modifications which can be readily figured out. □

Claim B.20. *Assuming* SSB *satisfies the 'index hiding' property, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,5)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.20 is analogous to that of Claim B.16 with certain approximate changes which are easy to determine. □

Claim B.21. *Assuming* ITR *satisfies the 'indistinguishability of enforcing setup' property, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,6)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,6)}(\lambda)|$ is non-negligible. Below, we construct a PPT adversary $\mathcal{B}$ that breaks the indistinguishability of enforcing setup property of the iterator ITR using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{U}_{\text{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r_{\text{SIG}}^* = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\text{SK}}_{\text{SIG}}^*, \widehat{\text{VK}}_{\text{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda;\ r_{\text{SIG}}^*)$.
  4. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$.
  3. For $j = 1, \ldots, \iota + 1$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
     - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
  4. Then, $\mathcal{B}$ sends $n_{\text{ITR}} = 2^\lambda$ along with the sequence of messages $((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0))$ to its ITR enforcing setup indistinguishability challenger $\mathcal{C}$ and receives back $(\text{PP}_{\text{ITR}}, v_0)$, where either $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$ or $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\text{ITR}} = 2^\lambda, ((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)))$.
  5. For $j = 1, \ldots, \iota + 1$, $\mathcal{B}$ iteratively computes $v_j = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{j-1}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$. It sets $m_{\iota,0}^{(\nu)} = (v_\iota, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$ and $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.
  6. It gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}, v_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, \\ \qquad h^*, \ell^*]) \end{array} \right).$$

- For $\theta \in [\hat{q}_{\text{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\text{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\text{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\text{SK}}_{\text{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\text{ABS}}^{(\theta)} = (\widehat{\text{VK}}_{\text{SIG}}^*, \sigma_{\text{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its ITR enforcing setup indistinguishability experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\text{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\text{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its ITR enforcing setup indistinguishability experiment.

Note that if $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$. On the other hand, if $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\text{ITR}} = 2^\lambda, ((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$. This completes the proof of Claim B.21. □

Claim B.22. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and ITR has the 'enforcing' property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,6)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,7)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota,7}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} signing key provided to $\mathcal{A}$, whereas, in $\mathsf{Hyb}_{0,\nu-1,3,\iota,7}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.11),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota')}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.8).

We will argue that the programs P_0 and P_1 are functionally identical, so that, by the security of $\mathcal{IO}$ Claim B.22 follows. The only inputs on which the outputs of the two programs can possibly differ are those corresponding to $(h, i) = (h^*, \iota)$. For inputs corresponding to (h^*, ι), the program P_0 checks whether '$m_{\text{IN}} = m_{\iota,0}^{(\nu)}$' and '$m_{\text{OUT}} = m_{\iota+1,0}^{(\nu)}$' to determine the type of the outputted signature, while the program P_1 only checks whether '$m_{\text{OUT}} = m_{\iota+1,0}^{(\nu)}$'. Thus, the two programs will be functionally equivalent if we can show that for inputs corresponding to (h^*, ι), $m_{\text{OUT}} = m_{\iota+1,0}^{(\nu)}$ implies $m_{\text{IN}} = m_{\iota,0}^{(\nu)}$. Recall that in both experiment $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)} \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\text{ITR}} = 2^\lambda, ((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)))$. Now, $m_{\text{OUT}} = m_{\iota+1,0}^{(\nu)}$ implies $v_{\text{OUT}} = v_{\iota+1}^{(\nu)}$. Therefore, by the enforcing property of ITR it follows that $v_{\text{IN}} = v_\iota^{(\nu)}$ and $(\text{ST}, w_{\text{IN}}, 0) = (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$, which in turn implies that $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0) = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0) = m_{\iota,0}^{(\nu)}$. □

Claim B.23. *Assuming* ITR *satisfies the 'indistinguishability of enforcing setup' property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,7)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,8)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim B.23 is analogous to that of Claim B.21 with some appropriate modifications which are easy to determine. □

□

Lemma B.6. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure punctureable pseudorandom function, and* SPS *is a splittable signature scheme satisfying '$\text{VK}_{\text{SPS-ONE}}$ indistinguishability', '$\text{VK}_{\text{SPS-ABO}}$ indistinguishability', as well as 'splitting indistinguishability', for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota')}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota+1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. In order to establish Lemma B.6, we consider the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota+1}$:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota+1}$

$\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$ except that in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$.
2. Then, it creates the punctured PPRF keys $K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},E}^{(\nu)}, (h^*, \iota+1))$ and $K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},F}^{(\nu)}, (h^*, \iota+1))$.

3. After that, it computes $r^{(\nu,\iota+1)}_{\text{SPS},G} = \mathcal{F}(K^{(\nu)}_{\text{SPS},E}, (h^*, \iota+1)), r^{(\nu,\iota+1)}_{\text{SPS},H} = \mathcal{F}(K^{(\nu)}_{\text{SPS},F}, (h^*, \iota+1))$, and forms $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},G}) = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,\iota+1)}_{\text{SPS},G}), (\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},H}) = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,\iota+1)}_{\text{SPS},H})$.
4. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
   - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
   - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
   - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
   - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

   It sets $m^{(\nu)}_{\iota+1,0} = (v^{(\nu)}_{\iota+1}, q^{(\nu)}_0, w^{(\nu)}_{\iota+1}, 0)$.
5. It gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\}, \\ \quad \text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, m^{(\nu)}_{\iota+1,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \\ \quad \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, \\ \quad h^*, \ell^*]) \end{array}\right),$$

   where the programs $\mathsf{Accumulate.Prog}^{(3,\iota',1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}$ respectively are the modifications of the programs $\mathsf{Accumulate.Prog}^{(3,\iota')}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}$ (Figs. A.8 and A.7) and are shown in Figs. B.12 and B.13.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$ with the only exception that while constructing the ν^{th} signing key queried by $\mathcal{A}$, $\mathcal{B}$ selects $r^{(\nu,\iota+1)}_{\text{SPS},G}, r^{(\nu,\iota+1)}_{\text{SPS},H} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, i.e., in other words, $\mathcal{B}$ generates $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},G}), (\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},H}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$ except that in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$.
2. Then, it creates the punctured PPRF keys $K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},E}, (h^*, \iota+1))$ and $K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},F}, (h^*, \iota+1))$.
3. After that it forms $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},G}), (\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},H}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$.
4. Next, it computes $m^{(\nu)}_{\iota+1,0} = (v^{(\nu)}_{\iota+1}, q^{(\nu)}_0, w^{(\nu)}_{\iota+1}, 0)$ just as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$.
5. After that, it creates $(\sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},G}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, m^{(\nu)}_{\iota+1,0})$ and $(\sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},H}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},H}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, m^{(\nu)}_{\iota+1,0})$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota+1)\}, K_{\text{SPS},F}\{(h^*, \iota+1)\}$, Signing keys SK_G, SK_H, Verification keys VK_G, VK_H, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h,i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$. Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
   (b) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h,i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$. Else, set $\text{VK}_{\text{SPS},F} = \text{VK}_H$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (0 \leq i \leq \iota) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$. Else, set $\text{SK}'_{\text{SPS},E} = \text{SK}_G$.
   (b) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h, i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$. Else, set $\text{SK}'_{\text{SPS},F} = \text{SK}_H$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. If $[(h,i) = (h^*, \iota)] \wedge [m_{\text{OUT}} = m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
   Else if $[(h,i) = (h^*, \iota)] \wedge [m_{\text{OUT}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
   Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. B.12. $\mathsf{Accumulate.Prog}^{(3,\iota',1)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota+1)\}$, $K_{\text{SPS},F}\{(h^*, \iota+1)\}$, Verification keys VK_G, VK_H, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) If $(h, \ell_{\text{INP}}) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$. Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
   (b) If $(h, \ell_{\text{INP}}) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$. Else, set $\text{VK}_{\text{SPS},F} = \text{VK}_H$.
   (c) Set $m = (v, \text{ST}, w, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =\text{'-'}] \wedge [(\ell_{\text{INP}} > \ell^*) \vee (0 < \ell_{\text{INP}} \leq \iota) \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. B.13. $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}$

6. It gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \underline{\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \\ \quad \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Accumulate.Prog}^{(3,\iota',2)}$ is an alteration of the program $\mathsf{Accumulate.Prog}^{(3,\iota',1)}$ (Fig. B.12) and is shown in Fig. B.14.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator $\mathrm{PP}_{\mathrm{ACC}}$, Public parameters for iterator $\mathrm{PP}_{\mathrm{ITR}}$, Punctured PPRF keys $K_{\mathrm{SPS},E}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}\{(h^*, \iota+1)\}$, Signature $\underline{\sigma_G}$, Signing key SK_H, Verification keys $\mathrm{VK}_G, \mathrm{VK}_H$, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol $\mathrm{SYM}_{\mathrm{IN}}$, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\mathrm{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) If $(h, i) \neq (h^*, \iota+1)$, compute $r_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}\{(h^*, \iota+1)\}, (h, i))$, $(\mathrm{SK}_{\mathrm{SPS},E}, \mathrm{VK}_{\mathrm{SPS},E}, \mathrm{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},E})$. Else, set $\mathrm{VK}_{\mathrm{SPS},E} = \mathrm{VK}_G$.
   (b) If $(h, i) \neq (h^*, \iota+1)$, compute $r_{\mathrm{SPS},F} = \mathcal{F}(K_{\mathrm{SPS},F}\{(h^*, \iota+1)\}, (h, i))$, $(\mathrm{SK}_{\mathrm{SPS},F}, \mathrm{VK}_{\mathrm{SPS},F}, \mathrm{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},F})$. Else, set $\mathrm{VK}_{\mathrm{SPS},F} = \mathrm{VK}_H$.
   (c) Set $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}, w_{\mathrm{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},E}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(i > \ell^*) \vee (0 \leq i \leq \iota) \vee (h \neq h^*)]$, output $\perp$. Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},F}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\mathrm{HK}, h, i, \mathrm{SYM}_{\mathrm{IN}}, \pi_{\mathrm{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, i, \mathrm{AUX})$. If $w_{\mathrm{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}}, (\mathrm{ST}, w_{\mathrm{IN}}, 0))$.
4. (a) If $(h, i) \neq (h^*, \iota)$, compute $r'_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}\{(h^*, \iota+1)\}, (h, i+1))$, $(\mathrm{SK}'_{\mathrm{SPS},E}, \mathrm{VK}'_{\mathrm{SPS},E}, \mathrm{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},E})$.
   (b) If $(h, i) \neq (h^*, \iota)$, compute $r'_{\mathrm{SPS},F} = \mathcal{F}(K_{\mathrm{SPS},F}\{(h^*, \iota+1)\}, (h, i+1))$, $(\mathrm{SK}'_{\mathrm{SPS},F}, \mathrm{VK}'_{\mathrm{SPS},F}, \mathrm{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},F})$. Else, set $\mathrm{SK}'_{\mathrm{SPS},F} = \mathrm{SK}_H$.
   (c) Set $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}, w_{\mathrm{OUT}}, 0)$. If $[(h, i) = (h^*, \iota)] \wedge [m_{\mathrm{OUT}} = m_{\iota+1,0}]$, $\underline{\text{set } \sigma_{\mathrm{SPS,OUT}} = \sigma_G}$.
   Else if $[(h, i) = (h^*, \iota)] \wedge [m_{\mathrm{OUT}} \neq m_{\iota+1,0}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},F}, m_{\mathrm{OUT}})$.
   Else if $i < \ell^*$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},\alpha}, m_{\mathrm{OUT}})$.
   Else, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},E}, m_{\mathrm{OUT}})$.
5. Output $(w_{\mathrm{OUT}}, v_{\mathrm{OUT}}, \sigma_{\mathrm{SPS,OUT}})$.

Fig. B.14. $\mathsf{Accumulate.Prog}^{(3,\iota',2)}$

$\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates everything as in $\mathsf{hyb}_{0,\nu-1,3,\iota',3}$, however, it hands $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \underline{\mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \underline{\mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}}, \\ \quad \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ forms all the components just as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$, however, it gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \underline{\mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \\ \quad \underline{\mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$, however, it returns the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\underline{\mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}}, \\ \quad \underline{\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*])}, \\ \mathcal{IO}(\underline{\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \\ \quad \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right)$$

to $\mathcal{A}$, where the program $\mathsf{Accumulate.Prog}^{(3,\iota',3)}$ is a modification of the program $\mathsf{Accumulate.Prog}^{(3,\iota',2)}$ (Fig. B.14) and is depicted in Fig. B.15. The remaining part of the experiment is identical to $\mathsf{hyb}_{0,\nu-1,3,\iota',5}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$ except that it does not generate $\underline{(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ONE},H}^{(\nu,\iota+1)}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)})}$ and provides $\mathcal{A}$ with the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \underline{\mathrm{SK}_{\text{SPS-ABO},G}^{(\nu,\iota+1)}}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \underline{\mathrm{VK}_{\text{SPS-ABO},G}^{(\nu,\iota+1)}}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \\ \quad \underline{\mathrm{VK}_{\text{SPS-ABO},G}^{(\nu,\iota+1)}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right).$$

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota{+}1)\}, K_{\text{SPS},F}\{(h^*, \iota{+}1)\}$, Signature σ_G, Signing key SK_H, Verification keys VK_G, VK_H, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h,i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$. Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
   (b) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h,i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$. Else, set $\text{VK}_{\text{SPS},F} = \text{VK}_H$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(i > \ell^*) \vee (0 \leq i \leq \iota) \vee (h \neq h^*)]$, output $\perp$. Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h, i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$. Else, set $\text{SK}'_{\text{SPS},F} = \text{SK}_H$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. If $[(h,i) = (h^*, \iota)] \wedge [m_{\text{OUT}} = m_{\iota+1,0}]$, set $\sigma_{\text{SPS,OUT}} = \sigma_G$.
   Else if $[(h,i) = (h^*, \iota)] \wedge [m_{\text{OUT}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
   Else if $[(h,i) = (h^*, \iota+1)] \wedge [m_{\text{IN}} = m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
   Else if $[(h,i) = (h^*, \iota+1)] \wedge [m_{\text{IN}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
   Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. B.15. $\mathsf{Accumulate.Prog}^{(3,\iota',3)}$

The rest of the experiment is the same as $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',8}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components as in $\mathsf{hyb}_{0,\nu-1,3,\iota',7}$, however, it returns the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',4)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \underline{\text{SK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*}]), \\ \mathcal{IO}(\underline{\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,2)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, h^*, \ell^*]}), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]}) \end{array}\right)$$

to $\mathcal{A}$, where the programs $\mathsf{Accumulate.Prog}^{(3,\iota',4)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,2)}$ respectively are the modifications of the programs $\mathsf{Accumulate.Prog}^{(3,\iota',3)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}$ (Figs. B.15 and B.13) and are shown in Figs. B.16 and B.17. The rest of the experiment if identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',9}$: This experiments analogous to $\mathsf{hyb}_{0,\nu-1,3,\iota',8}$ with the only exception that while constructing the ν^{th} signing key queried by $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates $\underline{(\text{SK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-REJ},G}^{(\nu,\iota+1)}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},G}^{(\nu,\iota+1)} = \mathcal{F}(K_{\text{SPS},E}^{(\nu)}, (h^*, \iota+1)))}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',10}$: This experiment corresponds to $\mathsf{hyb}_{0,\nu-1,3,\iota+1}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',\vartheta)}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,3,\iota',\vartheta}$,

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota{+}1)\}, K_{\text{SPS},F}\{(h^*, \iota{+}1)\}$, Signing key SK_G, Verification key VK_G, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1. (a) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,\iota+1)\}, (h,i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$. Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
   (b) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*,\iota+1)\}, (h,i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(i > \ell^*) \vee \underline{(0 \leq i \leq \iota+1)} \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4. (a) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,\iota+1)\}, (h,i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},E})$.
   (b) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*,\iota+1)\}, (h,i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},F})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. If $\underline{(h,i) = (h^*, \iota)}$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_G, m_{\text{OUT}})$.
   Else if $[(h,i) = (h^*, \iota+1)] \wedge [m_{\text{IN}} = m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
   Else if $[(h,i) = (h^*, \iota+1)] \wedge [m_{\text{IN}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
   Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. B.16. $\mathsf{Accumulate.Prog}^{(3,\iota',4)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*,\iota+1)\}$, $K_{\text{SPS},F}\{(h^*,\iota+1)\}$, Verification key VK_G, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) If $(h, \ell_{\text{INP}}) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,\iota+1)\}, (h,\ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
   (b) If $(h, \ell_{\text{INP}}) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*,\iota+1)\}, (h,\ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},F})$.
   (c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
   (e) If $[\alpha =$'-'$] \wedge [(\ell_{\text{INP}} > \ell^*) \vee \underline{(0 < \ell_{\text{INP}} \leq \iota+1)} \vee (h \neq h^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
   (f) If $\alpha =$'-', output $\perp$.
2. (a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{DSPS},B})$.
   (c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. B.17. $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,2)}$

for $\vartheta \in [0, 10]$. From the description of the hybrid experiments it follows that $\mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,3,\iota',0)}_{\mathcal{A}}(\lambda)$ and $\mathsf{Adv}^{(0,\nu-1,3,\iota+1)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,3,\iota',10)}_{\mathcal{A}}(\lambda)$. Hence, we have

$$|\mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota+1)}_{\mathcal{A}}(\lambda)| \leq \sum_{\vartheta=1}^{10} |\mathsf{Adv}^{(0,\nu-1,3,\iota',\vartheta-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',\vartheta)}_{\mathcal{A}}(\lambda)|. \tag{B.6}$$

Claims B.24–B.33 below will show that the RHS of Eq. (B.6) is negligible and thus Lemma B.6 follows.

Claim B.24. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$ and $\mathcal{F}$ satisfies the correctness under puncturing property, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,3,\iota',0)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',1)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P'_0)$ within the ν^{th} signing key provided to $\mathcal{A}$, while in $\mathsf{hyb}_{0,\nu-1,3,\iota',1}$, it includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(P'_1)$ instead, where

– $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota')}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, m^{(\nu)}_{\iota+1,0}, h^*, \ell^*]$ (Fig. A.8),

- $P_0' = \mathsf{Change\text{-}SPSProg}^{(3,\iota)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]$ (Fig. A.7),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota',1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, m^{(\nu)}_{\iota+1,0}, h^*, \ell^*]$ (Fig. B.12),
- $P_1' = \mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, h^*, \ell^*]$ (Fig. B.13).

Observe that by the correctness under puncturing property of the PPRF $\mathcal{F}$, the programs P_0 and P_1 are functionally identical for all inputs corresponding to $(h, i) \neq (h^*, \iota)$ and $(h, i) \neq (h^*, \iota+1)$. For inputs corresponding to (h^*, ι),the program P_1 uses the hardwired signing keys which are exactly same as those computed by the program P_0. The same is true for the hardwired verification keys used by P_1 for inputs corresponding to $(h^*, \iota+1)$. Thus, the programs P_0 and P_1 are functionally equivalent. A similar argument shows that the same is correct for programs P_0' and P_1'. Therefore, by the security of $\mathcal{IO}$ Claim B.24 follows. Ofcourse, we need to consider a sequence of intermediate hybrid experiments to switch the programs one at a time. □

Claim B.25. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,3,\iota',1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',2)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}^{(0,\nu-1,3,\iota',1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',2)}_{\mathcal{A}}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ is given below. We note that in the following we work in a model of selective pseudorandomness for PPRF involving two independent punctured keys and two challenge values for a challenge input, one under each key. However, this model is clearly equivalent to the original single punctured key and single challenge value model described in Definition 2.2 through a hybrid argument.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{U}_{\text{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r^*_{\text{SIG}} = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\text{SK}}^*_{\text{SIG}}, \widehat{\text{VK}}^*_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r^*_{\text{SIG}})$.
  4. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ sends $(h^*, \iota+1)$ as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back two punctured PPRF keys $K^*_1\{(h^*, \iota+1)\}, K^*_2\{(h^*, \iota+1)\}$ and two values $r^*_1, r^*_2 \in \mathcal{Y}_{\text{PPRF}}$, where either $r^*_1 = \mathcal{F}(K^*_1, (h^*, \iota+1)), r^*_2 = \mathcal{F}(K^*_2, (h^*, \iota+1))$ or $r^*_1, r^*_2 \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. $\mathcal{B}$ *implicitly* views the keys K^*_1 and K^*_2 as the keys $K^{(\nu)}_{\text{SPS},E}$ and $K^{(\nu)}_{\text{SPS},F}$ respectively.
  4. Next, $\mathcal{B}$ generates $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},G}) = \mathsf{SPS.Setup}(1^\lambda; r^*_1)$ and $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},H}) = \mathsf{SPS.Setup}(1^\lambda; r^*_2)$.
  5. After that, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
     - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
     - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
     - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
     - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

     It sets $m^{(\nu)}_{\iota+1,0} = (v^{(\nu)}_{\iota+1}, q^{(\nu)}_0, w^{(\nu)}_{\iota+1}, 0)$.

6. It gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_1^*\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_1^*\{(h^*, \iota+1)\}, K_2^*\{(h^*, \iota+1)\}, \\ \quad \mathrm{SK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{SK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_1^*\{(h^*, \iota+1)\}, K_2^*\{(h^*, \iota+1)\}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, \\ \quad h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\mathrm{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\mathrm{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}_{\mathrm{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\mathrm{ABS}}^{(\theta)} = (\widehat{\mathrm{VK}}_{\mathrm{SIG}}^*, \sigma_{\mathrm{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its PPRF selective pseudorandomness experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\mathrm{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its PPRF selective pseudorandomness experiment.

Note that if $r_1^* = \mathcal{F}(K_1^*, (h^*, \iota+1)), r_2^* = \mathcal{F}(K_2^*, (h^*, \iota+1))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$. On the other hand, if $r_1^*, r_2^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$, the $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$. This completes the proof of Claim B.25. □

Claim B.26. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',3)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} signing key provided to $\mathcal{A}$, while in $\mathsf{hyb}_{0,\nu-1,3,\iota',3}$, it includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota',1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{SK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{SK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.12),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.14).

Now, the only inputs on which the outputs of the two programs can possibly differ are those corresponding to $(h, i) = (h^*, \iota)$. However, observe that for inputs corresponding to (h^*, ι), if $m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$, then both programs clearly output the same signature, where P_0 computes the signature explicitly and P_1 has the signature hardwired into it. On the other hand, by the correctness [Property (ii)] of the splittable signature SPS defined in Definition 2.6 it follows that the programs P_0 and P_1 output same signatures even when $m_{\mathrm{OUT}} \neq m_{\iota+1,0}^{(\nu)}$ for inputs corresponding to (h^*, ι). Hence, the two programs are functionally equivalent. Therefore, Claim B.26 follows by the security of $\mathcal{IO}$. □

Claim B.27. *Assuming* SPS *is a splittable signature scheme satisfying '$\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability', for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',4)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',4)}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the $\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{U}_{\mathrm{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.

2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
3. After that, $\mathcal{B}$ computes $r^*_{\text{SIG}} = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\text{SK}}^*_{\text{SIG}}, \widehat{\text{VK}}^*_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r^*_{\text{SIG}})$.
4. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.

- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ creates the punctured PPRF keys $K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},E}, (h^*, \iota+1))$ and $K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},F}, (h^*, \iota+1))$.
  4. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
     - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
     - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
     - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
     - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

     It sets $m^{(\nu)}_{\iota+1,0} = (v^{(\nu)}_{\iota+1}, q^{(\nu)}_0, w^{(\nu)}_{\iota+1}, 0)$.
  5. After that, $\mathcal{B}$ sends $m^{(\nu)}_{\iota+1,0}$ as the challenge message to its SPS $\text{VK}_{\text{SPS-ONE}}$ indistinguishability challenger $\mathcal{C}$ and receives back a signature-verification key pair $(\sigma_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0}}, \text{VK})$, where VK is either a normal verification key VK_{SPS} or a one verification key $\text{VK}_{\text{SPS-ONE}}$ for the message $m^{(\nu)}_{\iota+1,0}$.
  6. Next, $\mathcal{B}$ generates $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},H}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$ and forms $(\sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0},H}$, $\text{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},H}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},H}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, m^{(\nu)}_{\iota+1,0})$.
  7. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}, \\ \quad K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\}, \sigma_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0}}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},H}, \text{VK}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, m^{(\nu)}_{\iota+1,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\}, \text{VK}, \\ \quad \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, \\ \quad K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\text{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\text{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma^{(\theta)}_{\text{SIG}} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\text{SK}}^*_{\text{SIG}}, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma^{(\theta)}_{\text{ABS}} = (\widehat{\text{VK}}^*_{\text{SIG}}, \sigma^{(\theta)}_{\text{SIG}})$.
- Finally, $\mathcal{A}$ output a signature σ^*_{ABS} on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its $\text{VK}_{\text{SPS-ONE}}$ indistinguishability experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\text{ABS}}) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\text{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its $\text{VK}_{\text{SPS-ONE}}$ indistinguishability experiment.

Notice that if $\text{VK} = \text{VK}_{\text{SPS}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$. On the other hand, if $\text{VK} = \text{VK}_{\text{SPS-ONE}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$. This completes the proof of Claim B.27. □

Claim B.28. *Assuming* SPS *is a splittable signature scheme satisfying* '$\text{VK}_{\text{SPS-ABO}}$ *indistinguishability', for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,3,\iota',4)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',5)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}^{(0,\nu-1,3,\iota',4)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',5)}_{\mathcal{A}}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the $\text{VK}_{\text{SPS-ABO}}$ indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r_{\text{SIG}}^* = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\text{SK}}_{\text{SIG}}^*, \widehat{\text{VK}}_{\text{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda;\ r_{\text{SIG}}^*)$.
  4. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \dots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ creates the punctured PPRF keys $K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},E}^{(\nu)}, (h^*, \iota+1))$ and $K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},F}^{(\nu)}, (h^*, \iota+1))$.
  4. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \dots, \iota+1$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
     - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
     - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

     It sets $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.
  5. After that, $\mathcal{B}$ sends $m_{\iota+1,0}^{(\nu)}$ as the challenge message to its SPS $\text{VK}_{\text{SPS-ABO}}$ indistinguishability challenger $\mathcal{C}$ and receives back an all-but-one signing key-verification key pair $(\text{SK}_{\text{SPS-ABO}}, \text{VK})$, where VK is either a normal verification key VK_{SPS} or an all-but-one verification key $\text{VK}_{\text{SPS-ABO}}$.
  6. Next, $\mathcal{B}$ generates $(\text{SK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-REJ},G}^{(\nu,\iota+1)}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$ and forms $(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \text{SK}_{\text{SPS-ABO},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-ABO},G}^{(\nu,\iota+1)}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS},G}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)})$.
  7. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \text{VK}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \text{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \\ \quad \text{VK}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \dots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \\ \quad K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\text{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\text{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\text{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\text{SK}}_{\text{SIG}}^*, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\text{ABS}}^{(\theta)} = (\widehat{\text{VK}}_{\text{SIG}}^*, \sigma_{\text{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ_{ABS}^* on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its $\text{VK}_{\text{SPS-ABO}}$ indistinguishability experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\text{ABS}}^*) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\text{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its $\text{VK}_{\text{SPS-ABO}}$ indistinguishability experiment.

Notice that if $\text{VK} = \text{VK}_{\text{SPS}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$. On the other hand, if $\text{VK} = \text{VK}_{\text{SPS-ABO}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$. This completes the proof of Claim B.28. □

Claim B.29. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',6)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} signing key returned to $\mathcal{A}$, while in $\mathsf{hyb}_{0,\nu-1,3,\iota',6}$, it includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \text{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.14),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \text{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.15).

We will argue that the programs P_0 and P_1 are functionally equivalent, so that, by the security of $\mathcal{IO}$ Claim B.29 holds. First of all observe that the constants hardwired in both the programs are identically generated. Clearly, the inputs on which the outputs of the programs P_0 and P_1 can possibly differ are those corresponding to $(h, i) = (h^*, \iota+1)$. For inputs corresponding to $(h^*, \iota+1)$, let us consider the following two cases:

(I) ($m_{\text{IN}} = m_{\iota+1,0}^{(\nu)}$): In this case, using the correctness [Properties (i), (iii) and (vi)] of the splittable signature SPS described in Definition 2.6 it follows that for both programs either $\alpha =$'-' or $\alpha =$'E'. Now, if $\alpha =$'-', then both programs output $\perp$. On the other hand, if $\alpha =$'E', then P_0 outputs the signature $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}}) = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$, which is the same signature that P_1 is programmed to output in this case. Thus, both programs have identical outputs in this case.

(II) ($m_{\text{IN}} \neq m_{\iota+1,0}^{(\nu)}$): In this case, we use the correctness [Property (v)] of SPS described in Definition 2.6 to conclude that $\alpha \neq$'E' and correctness [Properties (i) and (iv)] of SPS confirms that either $\alpha =$'-' or $\alpha =$'F'. Now, if $\alpha =$'-', then both programs output $\perp$ as earlier. Otherwise, if $\alpha =$'F', then P_0 outputs $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}}) = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$, which P_1 is programmed to output in this case. Therefore, both programs are functionally equivalent in this case as well.

□

Claim B.30. *Assuming* SPS *is a splittable signature scheme satisfying 'splitting indistinguishability', for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',6)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',7)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',6)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',7)}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the splitting indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r_{\text{SIG}}^* = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\text{SK}}_{\text{SIG}}^*, \widehat{\text{VK}}_{\text{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}}^*)$.
  4. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ creates the punctured PPRF keys $K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},E}^{(\nu)}, (h^*, \iota+1))$ and $K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},F}^{(\nu)}, (h^*, \iota+1))$.
  4. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
     - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$

– $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

It sets $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.

5. After that, $\mathcal{B}$ sends $m_{\iota+1,0}^{(\nu)}$ as the challenge message to its SPS splitting indistinguishability challenger $\mathcal{C}$ and receives back a tuple $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}^*_{\text{SPS-ONE}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ABO}})$, where
   – either $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}^*_{\text{SPS-ONE}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ABO}}) =$
   $(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}_{\text{SPS-ONE}}, \mathrm{SK}_{\text{SPS-ABO}}, \mathrm{VK}_{\text{SPS-ABO}})$
   – or $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}^*_{\text{SPS-ONE}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ABO}}) =$
   $(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}_{\text{SPS-ONE}}, \mathrm{SK}'_{\text{SPS-ABO}}, \mathrm{VK}'_{\text{SPS-ABO}})$

   such that $(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}_{\text{SPS-ONE}}, \mathrm{SK}_{\text{SPS-ABO}}, \mathrm{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}_{\mathrm{SPS}}, m_{\iota+1,0}^{(\nu)}), (\sigma'_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}'_{\text{SPS-ONE}}, \mathrm{SK}'_{\text{SPS-ABO}}, \mathrm{VK}'_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}'_{\mathrm{SPS}}, m_{\iota+1,0}^{(\nu)})$, $\mathrm{SK}_{\mathrm{SPS}}$ and $\mathrm{SK}'_{\mathrm{SPS}}$ being two independently generated signing keys for SPS.
6. $\mathcal{B}$ gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ONE},}, \mathrm{VK}^*_{\text{SPS-ABO}}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{VK}^*_{\text{SPS-ONE}}, \\ \quad \mathrm{VK}^*_{\text{SPS-ABO}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\mathrm{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\mathrm{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\mathrm{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\mathrm{ABS}}^{(\theta)} = (\widehat{\mathrm{VK}}^*_{\mathrm{SIG}}, \sigma_{\mathrm{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ^*_{ABS} on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its SPS splitting indistinguishability experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\mathrm{ABS}}) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its SPS splitting indistinguishability experiment.

Notice that if $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}^*_{\text{SPS-ONE}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ABO}}) = (\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}_{\text{SPS-ONE}}, \mathrm{SK}'_{\text{SPS-ABO}}, \mathrm{VK}'_{\text{SPS-ABO}})$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$. On the other hand, if $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}^*_{\text{SPS-ONE}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ABO}}) = (\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}_{\text{SPS-ONE}}, \mathrm{SK}_{\text{SPS-ABO}}, \mathrm{VK}_{\text{SPS-ABO}})$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$. This completes the proof of Claim B.30. □

Claim B.31. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',7)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',8)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota',8}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P'_0)$ within the ν^{th} signing key returned to $\mathcal{A}$, while in $\mathsf{hyb}_{0,\nu-1,3,\iota',8}$, it includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(P'_1)$ instead, where

– $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \mathrm{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.15),
– $P'_0 = \mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{VK}^{(\nu,\iota=1)}_{\text{SPS-ONE},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}, h^*, \ell^*]$ (Fig. B.13),
– $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota',4)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{SK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. B.16),
– $P'_1 = \mathsf{Change\text{-}SPS.Prog}^{(3,\iota,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{VK}^{(\nu,\iota=1)}_{\mathrm{SPS},G}, h^*, \ell^*]$ (Fig. B.17).

We will argue that the programs P_0 and P_1, as well as , the programs P_0' and P_1' are functionally equivalent, so that, by the security of $\mathcal{IO}$ Claim B.31 follows. First consider the programs P_0 and P_1. Clearly the only inputs on which the outputs of the two programs can possibly differ are those corresponding to $(h,i) = (h^*, \iota)$ and $(h,i) = (h^*, \iota+1)$. Now, for inputs corresponding to (h^*, ι), the outputs of the two programs are identical due to the correctness [Property (ii)] of the splittable signature SPS described in Definition 2.6 and the fact that the hardwired signature $\sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0},G}$ and the all-but-one signing key $\text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}$ used by the program P_0 for inputs corresponding to (h^*, ι) are obtained by running $\mathsf{SPS.Split}(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, m^{(\nu)}_{\iota+1,0})$, while the hardwired signing key used by P_1 in this case is $\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}$. Similarly, for inputs corresponding to $(h^*, \iota+1)$, the outputs of the two programs are also identical because of the correctness [Properties (i), (iii), (v), (iv) and (vi)] of SPS and the fact that the hardwired one and all-but-one verification keys used by the program P_0 for inputs corresponding to $(h^*, \iota+1)$ are generated by running $\mathsf{SPS.Split}(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, m^{(\nu)}_{\iota+1,0})$, while the hardwired verification key used by the program P_1 in this case is $\text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}$, which is the matching verification key of $\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}$. Hence, the two programs are functionally equivalent. The same type of argument holds for the programs P_0' and P_1'. □

Claim B.32. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,3,\iota',8)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',9)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim B.32 is analogous to that of Claim B.25 with some appropriate modifications that are readily identifiable. □

Claim B.33. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,3,\iota',9)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',10)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim B.33 is similar to that of Claim B.24 with some appropriate modifications which are easy to find out. □

□

Lemma B.7. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function, and* SPS *is a secure splittable signature scheme satisfying '$\text{VK}_{\text{SPS-ONE}}$ indistinguishability', '$\text{VK}_{\text{SPS-ABO}}$ indistinguishability', as well as 'splitting indistinguishability', for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,3,(\ell^*-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Lemma B.7 is similar to that of Lemma B.6 with certain appropriate changes which can be readily determined. □

Lemma B.8. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function, ACC is a secure positional accumulator possessing the 'indistinguishability of read setup' as well as 'read enforcing', and* SPS *is a secure splittable signature scheme satisfying '$\text{VK}_{\text{SPS-ONE}}$ indistinguishability', '$\text{VK}_{\text{SPS-ABO}}$ indistinguishability', as well as 'splitting indistinguishability', for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,0')}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. In order to establish Lemma B.8, we consider the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,4}$ and $\mathsf{Hyb}_{0,\nu-1,4,0'}$:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1,4}$ and $\mathsf{Hyb}_{0,\nu-1,4,0'}$

$\mathsf{Hyb}_{0,\nu-1,4\text{-I}}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,4}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-II}}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,4\text{-I}}$ except that in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,4\text{-I}}$.

2. Then, it creates the punctured PPRF keys $K^{(\nu)}_{\text{SPS},A}\{(h^*,\ell^*,0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},A},(h^*,\ell^*,0))$ and $K^{(\nu)}_{\text{SPS},B}\{(h^*,\ell^*,0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},B},(h^*,\ell^*,0))$.
3. After that, it computes $r^{(\nu,0)}_{\text{SPS},C} = \mathcal{F}(K^{(\nu)}_{\text{SPS},A},(h^*,\ell^*,0)), r^{(\nu,0)}_{\text{SPS},D} = \mathcal{F}(K^{(\nu)}_{\text{SPS},B},(h^*,\ell^*,0))$, and forms $(\text{SK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},C}) = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,0)}_{\text{SPS},C}), (\text{SK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},D}) = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,0)}_{\text{SPS},D})$.
4. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1,\ldots,\ell^*$, it iteratively computes the following:
   - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
   - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
   - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
   - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

   It sets $m^{(\nu)}_{\ell^*,0} = (v^{(\nu)}_{\ell^*}, q^{(\nu)}_0, w^{(\nu)}_{\ell^*}, 0)$.
5. It gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,1)}[K^{(\nu)}_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},E}, \text{SK}^{(\nu,0)}_{\text{SPS},C}, \text{SK}^{(\nu,0)}_{\text{SPS},D}, m^{(\nu)}_{\ell^*,0}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, \\ \quad K^{(\nu)}_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*,\ell^*,0)\}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Change\text{-}SPS.Prog}^{(4,1)}$ and $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\text{ABS}}$ respectively are the modifications of the programs $\mathsf{Change\text{-}SPS.Prog}^{(4)}$ and $\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{ABS}}$ (Figs. A.9 and A.1) and are depicted in Figs. B.18 and B.19.

Constants: Punctured PPRF keys $K_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K_{\text{SPS},B}\{(h^*,\ell^*,0)\}$, PPRF key $K_{\text{SPS},E}$, Signing keys SK_C, SK_D, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E},(h,\ell_{\text{INP}}), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Set $m = (v, \text{ST}, w, 0)$.
   (c) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2. (a) If $(h,\ell_{\text{INP}}) \neq (h^*,\ell^*)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*,\ell^*,0)\},(h,\ell_{\text{INP}},0)), (\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   Else, set $\text{SK}_{\text{SPS},A} = \text{SK}_C$.
   (b) If $(h,\ell_{\text{INP}}) \neq (h^*,\ell^*)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*,\ell^*,0)\},(h,\ell_{\text{INP}},0)), (\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   Else, set $\text{SK}_{\text{SPS},B} = \text{SK}_D$.
   (c) If $[(h,\ell_{\text{INP}}) = (h^*,\ell^*)] \wedge [m \neq m_{\ell^*,0}]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. B.18. $\mathsf{Change\text{-}SPS.Prog}^{(4,1)}$

$\mathsf{Hyb}_{0,\nu-1,4\text{-III}}$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-II}}$ with the only exception that while constructing the ν^{th} signing key queried by $\mathcal{A}$, $\mathcal{B}$ selects $r^{(\nu,0)}_{\text{SPS},C}, r^{(\nu,0)}_{\text{SPS},D} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, i.e., in other words, $\mathcal{B}$ generates $(\text{SK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},C}), (\text{SK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},D}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-IV}}$: This experiment is similar to $\mathsf{Hyb}_{0,\nu-1,4\text{-III}}$ except that in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,4\text{-III}}$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda$, Punctured PPRF keys $K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}$, Verification keys VK_C, VK_D, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or (Header Position POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3. (a) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   Else, set $\text{VK}_{\text{SPS},A} = \text{VK}_C$.
   (b) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   Else, set $\text{VK}_{\text{SPS},B} = \text{VK}_D$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$ '-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$ 'A'.
   (e) If $[\alpha =$ '-'$] \wedge [(t > t^*) \vee (h \neq h^*) \vee (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$ '-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$ 'B'.
   (f) If $\alpha =$ '-', output $\perp$.
4. (a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$ 'B'$]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6. (a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. B.19. $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\text{ABS}}$

2. Then, it creates the punctured PPRF keys $K^{(\nu)}_{\text{SPS},A}\{(h^*, \ell^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},A}, (h^*, \ell^*, 0))$ and $K^{(\nu)}_{\text{SPS},B}\{(h^*, \ell^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},B}, (h^*, \ell^*, 0))$.
3. After that it forms $(\text{SK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},C}), (\text{SK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},D}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$.
4. Next, it computes $m^{(\nu)}_{\ell^*,0} = (v^{(\nu)}_{\ell^*}, q^{(\nu)}_0, w^{(\nu)}_{\ell^*}, 0)$ just as in $\mathsf{Hyb}_{0,\nu-1,4\text{-III}}$.
5. After that, it creates $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},C}, \text{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \text{SK}^{(\nu,0)}_{\text{SPS-ABO},C}, \text{VK}^{(\nu,0)}_{\text{SPS-ABO},C}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,0)}_{\text{SPS},C}, m^{(\nu)}_{\ell^*,0})$ and $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},D}, \text{VK}^{(\nu,0)}_{\text{SPS-ONE},D}, \text{SK}^{(\nu,0)}_{\text{SPS-ABO},D}, \text{VK}^{(\nu,0)}_{\text{SPS-ABO},D}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,0)}_{\text{SPS},D}, m^{(\nu)}_{\ell^*,0})$.
6. It gives $\mathcal{A}$ the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K^{(\nu)}_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, K^{(\nu)}_{\text{SPS},E}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},C}, \\ \quad \text{SK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\text{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, \\ \quad K^{(\nu)}_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, h^*, \ell^*]) \end{array} \right),$$

where the program $\mathsf{Change\text{-}SPS.Prog}^{(4,2)}$ is an alteration of the program $\mathsf{Change\text{-}SPS.Prog}^{(4,1)}$ (Fig. B.18) and is shown in Fig. B.20.

$\mathsf{Hyb}_{0,\nu-1,4\text{-V}}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components as in $\mathsf{hyb}_{0,\nu-1,4\text{-IV}}$, however, it hands $\mathcal{A}$

Constants: Punctured PPRF keys $K_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K_{\text{SPS},B}\{(h^*,\ell^*,0)\}$, PPRF key $K_{\text{SPS},E}$, Signature σ_C, Signing key SK_D, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E},(h,\ell_{\text{INP}}),(\text{SK}_{\text{SPS},E},\text{VK}_{\text{SPS},E},\text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},E})$.
   (b) Set $m = (v, \text{ST}, w, 0)$.
   (c) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2. (a) If $(h,\ell_{\text{INP}}) \neq (h^*,\ell^*)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*,\ell^*,0)\},(h,\ell_{\text{INP}},0)),(\text{SK}_{\text{SPS},A},\text{VK}_{\text{SPS},A},\text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   (b) If $(h,\ell_{\text{INP}}) \neq (h^*,\ell^*)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*,\ell^*,0)\},(h,\ell_{\text{INP}},0)),(\text{SK}_{\text{SPS},B},\text{VK}_{\text{SPS},B},\text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   Else, set $\text{SK}_{\text{SPS},B} = \text{SK}_D$.
   (c) If $[(h,\ell_{\text{INP}}) = (h^*,\ell^*)] \wedge [m \neq m_{\ell^*,0}]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
   Else if $[(h,\ell_{\text{INP}} = (h^*,\ell^*)] \wedge [m = m_{\ell^*,0}]$, output $\sigma_{\text{SPS,OUT}} = \sigma_C$.
   Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. B.20. $\mathsf{Change\text{-}SPS.Prog}^{(4,2)}$

the signing key

$$\text{SK}_{\text{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\text{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\text{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\text{SPS},E}^{(\nu)}, \sigma_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}^{(\nu,0)}, \\ \quad \text{SK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1,0,1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\text{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\text{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, \underline{\text{VK}_{\text{SPS-ONE},C}^{(\nu,0)}}, \text{VK}_{\text{SPS},D}^{(\nu,0)}, h^*, \ell^*]) \end{array}\right),$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-IV}}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-VI}}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{hyb}_{0,\nu-1,4\text{-V}}$, however, it hands $\mathcal{A}$ the signing key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\text{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\text{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\text{SPS},E}^{(\nu)}, \sigma_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}^{(\nu,0)}, \\ \quad \text{SK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1,0,1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\text{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\text{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, \text{VK}_{\text{SPS-ONE},C}^{(\nu,0)}, \underline{\text{VK}_{\text{SPS-ABO},D}^{(\nu,0)}}, h^*, \ell^*]) \end{array}\right),$$

The rest of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,4\text{-V}}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-VII}}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components just as in $\mathsf{Hyb}_{0,\nu-1,4\text{-VI}}$, but it provides

$\mathcal{A}$ with the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\mathrm{SSB\text{-}BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \sigma^{(\nu,0)}_{\mathrm{SPS\text{-}ONE},m^{(\nu)}_{\ell^*,0},C}, \\ \quad \mathrm{SK}^{(\nu,0)}_{\mathrm{SPS\text{-}ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}^{(1,0,2)}_{\mathrm{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)},} \\ \quad \underline{K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, \mathrm{VK}^{(\nu,0)}_{\mathrm{SPS\text{-}ONE},C}, \mathrm{VK}^{(\nu,0)}_{\mathrm{SPS\text{-}ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*])} \end{array} \right),$$

where the program $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,2)}_{\mathrm{ABS}}$ is a modification of the program $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\mathrm{ABS}}$ (Fig. B.19) and is shown in Fig. B.21. The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-VI}}$.

Constants: TM $M = \langle Q, \Sigma_{\mathrm{INP}}, \Sigma_{\mathrm{TAPE}}, \delta, q_0, q_{\mathrm{AC}}, q_{\mathrm{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator $\mathrm{PP}_{\mathrm{ACC}}$, Public parameters for iterator $\mathrm{PP}_{\mathrm{ITR}}$, PPRF keys $K, K_1, \ldots, K_\lambda$, Punctured PPRF keys $K_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}$, Verification keys $\mathrm{VK}_C, \mathrm{VK}_D$, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String $\mathrm{SEED}_{\mathrm{IN}}$, Header position $\mathrm{POS}_{\mathrm{IN}}$, Symbol $\mathrm{SYM}_{\mathrm{IN}}$, TM state $\mathrm{ST}_{\mathrm{IN}}$, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\mathrm{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K,(h,\ell_{\mathrm{INP}}))$, or (Header Position $\mathrm{POS}_{\mathrm{OUT}}$, Symbol $\mathrm{SYM}_{\mathrm{OUT}}$, TM state $\mathrm{ST}_{\mathrm{OUT}}$, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\mathrm{SPS,OUT}}$, String $\mathrm{SEED}_{\mathrm{OUT}}$), or $\perp$

1. Identify an integer τ such that $2^\tau \le t < 2^{\tau+1}$. If $[\mathsf{PRG}(\mathrm{SEED}_{\mathrm{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau,(h,\ell_{\mathrm{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}, \pi_{\mathrm{ACC}}) = 0$, output $\perp$.
3. (a) If $(h,\ell_{\mathrm{INP}},t) \neq (h^*,\ell^*,1)$, compute $r_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\},(h,\ell_{\mathrm{INP}},t-1)), (\mathrm{SK}_{\mathrm{SPS},A}, \mathrm{VK}_{\mathrm{SPS},A}, \mathrm{VK}_{\mathrm{SPS\text{-}REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},A})$.
   Else, set $\mathrm{VK}_{\mathrm{SPS},A} = \mathrm{VK}_C$.
   (b) If $(h,\ell_{\mathrm{INP}},t) \neq (h^*,\ell^*,1)$, compute $r_{\mathrm{SPS},B} = \mathcal{F}(K_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\},(h,\ell_{\mathrm{INP}},t-1)), (\mathrm{SK}_{\mathrm{SPS},B}, \mathrm{VK}_{\mathrm{SPS},B}, \mathrm{VK}_{\mathrm{SPS\text{-}REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},B})$.
   Else, set $\mathrm{VK}_{\mathrm{SPS},B} = \mathrm{VK}_D$.
   (c) Set $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}_{\mathrm{IN}}, w_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},A}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \wedge [(t > t^*) \vee (h \neq h^*) \vee (\ell_{\mathrm{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},B}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4. (a) Compute $(\mathrm{ST}_{\mathrm{OUT}}, \mathrm{SYM}_{\mathrm{OUT}}, \beta) = \delta(\mathrm{ST}_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}})$ and $\mathrm{POS}_{\mathrm{OUT}} = \mathrm{POS}_{\mathrm{IN}} + \beta$.
   (b) If $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{REJ}}$, output $\perp$.
   Else if $[\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}] \wedge [\alpha =$'$B$'$]$, output $\perp$.
   Else if $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}$, perform the following:
   (I) Compute $r_{\mathrm{SIG}} = \mathcal{F}(K,(h,\ell_{\mathrm{INP}}))$, $(\mathrm{SK}_{\mathrm{SIG}}, \mathrm{VK}_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\mathrm{SIG}})$.
   (II) Output $(\mathrm{SK}_{\mathrm{SIG}}, \mathrm{VK}_{\mathrm{SIG}})$.
5. (a) Compute $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{OUT}}, \mathrm{POS}_{\mathrm{IN}}, \mathrm{AUX})$. If $w_{\mathrm{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}}, (\mathrm{ST}_{\mathrm{IN}}, w_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}))$.
6. (a) Compute $r'_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\},(h,\ell_{\mathrm{INP}},t)), (\mathrm{SK}'_{\mathrm{SPS},A}, \mathrm{VK}'_{\mathrm{SPS},A}, \mathrm{VK}'_{\mathrm{SPS\text{-}REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},A})$.
   (b) Compute $r'_{\mathrm{SPS},B} = \mathcal{F}(K_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\},(h,\ell_{\mathrm{INP}},t)), (\mathrm{SK}'_{\mathrm{SPS},B}, \mathrm{VK}'_{\mathrm{SPS},B}, \mathrm{VK}'_{\mathrm{SPS\text{-}REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},B})$.
   (c) Set $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}_{\mathrm{OUT}}, w_{\mathrm{OUT}}, \mathrm{POS}_{\mathrm{OUT}})$.
   If $[(h,\ell_{\mathrm{INP}},t) = (h^*,\ell^*,1)] \wedge [m_{\mathrm{IN}} = m_{\ell^*,0}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},A}, m_{\mathrm{OUT}})$.
   Else if $[(h,\ell_{\mathrm{INP}},t) = (h^*,\ell^*,1)] \wedge [m_{\mathrm{IN}} \neq m_{\ell^*,0}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},B}, m_{\mathrm{OUT}})$.
   Else, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},\alpha}, m_{\mathrm{OUT}})$.
7. If $t+1 = 2^{\tau'}$, for some integer τ', set $\mathrm{SEED}_{\mathrm{OUT}} = \mathcal{F}(K_{\tau'},(h,\ell_{\mathrm{INP}}))$.
   Else, set $\mathrm{SEED}_{\mathrm{OUT}} = \epsilon$.
8. Output $(\mathrm{POS}_{\mathrm{OUT}}, \mathrm{SYM}_{\mathrm{OUT}}, \mathrm{ST}_{\mathrm{OUT}}, w_{\mathrm{OUT}}, v_{\mathrm{OUT}}, \sigma_{\mathrm{SPS,OUT}}, \mathrm{SEED}_{\mathrm{OUT}})$.

Fig. B.21. $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,2)}_{\mathrm{ABS}}$

$\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$: In This experiment is the same as $\mathsf{Hyb}_{0,\nu-1,4\text{-VII}}$ with the only exception that while creating the ν^{th} signing key queried by $\mathcal{A}$, $\mathcal{B}$ generates $\underline{(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\mathrm{ACC\text{-}BLK}} = 2^\lambda, ((x_0^*,0),\ldots,(x_{\ell^*-1}^*,\ell^*-1)), i^* = 0)}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-IX}}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components just as in $\mathsf{hyb}_{0,\nu-1,4\text{-VIII}}$, however, it gives

$\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \sigma_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}^{(\nu,0)}, \\ \quad \mathrm{SK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1,0,3)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, \mathrm{VK}_{\text{SPS-ONE},C}^{(\nu,0)}, \mathrm{VK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]}) \end{array}\right),$$

where the program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1,0,3)}$ is a modification of the program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1,0,2)}$ (Fig. B.22) and is shown in Fig. B.22. The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$.

Constants: TM $M = \langle Q, \Sigma_{\mathrm{INP}}, \Sigma_{\mathrm{TAPE}}, \delta, q_0, q_{\mathrm{AC}}, q_{\mathrm{REJ}}\rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator $\mathrm{PP}_{\mathrm{ACC}}$, Public parameters for iterator $\mathrm{PP}_{\mathrm{ITR}}$, PPRF keys $K, K_1, \ldots, K_\lambda$, Punctured PPRF keys $K_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}$, Verification keys $\mathrm{VK}_C, \mathrm{VK}_D$, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String $\mathrm{SEED}_{\mathrm{IN}}$, Header position $\mathrm{POS}_{\mathrm{IN}}$, Symbol $\mathrm{SYM}_{\mathrm{IN}}$, TM state $\mathrm{ST}_{\mathrm{IN}}$, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\mathrm{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K,(h,\ell_{\mathrm{INP}}))$, or (Header Position $\mathrm{POS}_{\mathrm{OUT}}$, Symbol $\mathrm{SYM}_{\mathrm{OUT}}$, TM state $\mathrm{ST}_{\mathrm{OUT}}$, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\mathrm{SPS,OUT}}$, String $\mathrm{SEED}_{\mathrm{OUT}}$), or $\perp$

1. Identify an integer τ such that $2^\tau \le t < 2^{\tau+1}$. If $[\mathsf{PRG}(\mathrm{SEED}_{\mathrm{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau,(h,\ell_{\mathrm{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}, \pi_{\mathrm{ACC}}) = 0$, output $\perp$.
3. (a) If $(h,\ell_{\mathrm{INP}},t) \neq (h^*,\ell^*,1)$, compute $r_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\},(h,\ell_{\mathrm{INP}},t-1)), (\mathrm{SK}_{\mathrm{SPS},A}, \mathrm{VK}_{\mathrm{SPS},A}, \mathrm{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},A})$.
   Else, set $\mathrm{VK}_{\mathrm{SPS},A} = \mathrm{VK}_C$.
   (b) If $(h,\ell_{\mathrm{INP}},t) \neq (h^*,\ell^*,1)$, compute $r_{\mathrm{SPS},B} = \mathcal{F}(K_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\},(h,\ell_{\mathrm{INP}},t-1)), (\mathrm{SK}_{\mathrm{SPS},B}, \mathrm{VK}_{\mathrm{SPS},B}, \mathrm{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},B})$.
   Else, set $\mathrm{VK}_{\mathrm{SPS},B} = \mathrm{VK}_D$.
   (c) Set $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}_{\mathrm{IN}}, w_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},A}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \wedge [(t > t^*) \vee (h \neq h^*) \vee (\ell_{\mathrm{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},B}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4. (a) Compute $(\mathrm{ST}_{\mathrm{OUT}}, \mathrm{SYM}_{\mathrm{OUT}}, \beta) = \delta(\mathrm{ST}_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}})$ and $\mathrm{POS}_{\mathrm{OUT}} = \mathrm{POS}_{\mathrm{IN}} + \beta$.
   (b) If $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{REJ}}$, output $\perp$.
   Else if $[\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}] \wedge [\alpha =$'$B$'$]$, output $\perp$.
   Else if $[\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}] \wedge [\alpha =$'$A$'$] \wedge [(h,\ell_{\mathrm{INP}}) = (h^*,\ell^*)] \wedge [t \le 1]$, output $\perp$.
   Else if $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}$, perform the following:
   (I) Compute $r_{\mathrm{SIG}} = \mathcal{F}(K,(h,\ell_{\mathrm{INP}}))$, $(\mathrm{SK}_{\mathrm{SIG}}, \mathrm{VK}_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\mathrm{SIG}})$.
   (II) Output $(\mathrm{SK}_{\mathrm{SIG}}, \mathrm{VK}_{\mathrm{SIG}})$.
5. (a) Compute $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{OUT}}, \mathrm{POS}_{\mathrm{IN}}, \mathrm{AUX})$. If $w_{\mathrm{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}}, (\mathrm{ST}_{\mathrm{IN}}, w_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}))$.
6. (a) Compute $r'_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\},(h,\ell_{\mathrm{INP}},t)), (\mathrm{SK}'_{\mathrm{SPS},A}, \mathrm{VK}'_{\mathrm{SPS},A}, \mathrm{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},A})$.
   (b) Compute $r'_{\mathrm{SPS},B} = \mathcal{F}(K_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\},(h,\ell_{\mathrm{INP}},t)), (\mathrm{SK}'_{\mathrm{SPS},B}, \mathrm{VK}'_{\mathrm{SPS},B}, \mathrm{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},B})$.
   (c) Set $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}_{\mathrm{OUT}}, w_{\mathrm{OUT}}, \mathrm{POS}_{\mathrm{OUT}})$.
   If $[(h,\ell_{\mathrm{INP}},t) = (h^*,\ell^*,1)] \wedge [m_{\mathrm{IN}} = m_{\ell^*,0}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},A}, m_{\mathrm{OUT}})$.
   Else if $[(h,\ell_{\mathrm{INP}},t) = (h^*,\ell^*,1)] \wedge [m_{\mathrm{IN}} \neq m_{\ell^*,0}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},B}, m_{\mathrm{OUT}})$.
   Else, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},\alpha}, m_{\mathrm{OUT}})$.
7. If $t+1 = 2^{\tau'}$, for some integer τ', set $\mathrm{SEED}_{\mathrm{OUT}} = \mathcal{F}(K_{\tau'},(h,\ell_{\mathrm{INP}}))$.
   Else, set $\mathrm{SEED}_{\mathrm{OUT}} = \epsilon$.
8. Output $(\mathrm{POS}_{\mathrm{OUT}}, \mathrm{SYM}_{\mathrm{OUT}}, \mathrm{ST}_{\mathrm{OUT}}, w_{\mathrm{OUT}}, v_{\mathrm{OUT}}, \sigma_{\mathrm{SPS,OUT}}, \mathrm{SEED}_{\mathrm{OUT}})$.

Fig. B.22. $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1,0,3)}$

$\mathsf{hyb}_{0,\nu-1,4\text{-X}}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,4\text{-IX}}$ with the only exception that while constructing the ν^{th} signing key queried by $\mathcal{A}$, $\mathcal{B}$ forms $\underline{(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-XI}}$: In this experiment, in response to the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,4\text{-X}}$ except that it does not generate $(\sigma_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},D}^{(\nu,0)}, \mathrm{VK}_{\text{SPS-ONE},D}^{(\nu,0)}, \mathrm{SK}_{\text{SPS-ABO},D}^{\nu,0)}, \mathrm{VK}_{\text{SPS-ABO},D}^{(\nu,0)}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}_{\mathrm{SPS},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)})$ and

hands $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \sigma_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}^{(\nu,0)}, \\ \quad \underline{\mathrm{SK}_{\text{SPS-ABO},C}^{(\nu,0)}}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1,0,3)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, \mathrm{VK}_{\text{SPS-ONE},C}^{(\nu,0)}, \underline{\mathrm{VK}_{\text{SPS-ABO},C}^{(\nu,0)}}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-X}}$.

$\mathsf{hyb}_{0,\nu-1,4\text{-XII}}$: In this experiment, to answer the ν^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,4\text{-XI}}$, but it provides $\mathcal{A}$ with the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\underline{\mathsf{Change\text{-}SPS.Prog}^{(4,3)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \mathrm{SK}_{\mathrm{SPS},C}^{(\nu,0)}, h^*, \ell^*]}), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1,0,4)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, \mathrm{VK}_{\mathrm{SPS},C}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]}) \end{array}\right),$$

where the programs $\mathsf{Change\text{-}SPS.Prog}^{(4,3)}$ and $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1,0,4)}$ are the alterations of the programs $\mathsf{Change\text{-}SPS.Prog}^{(4,2)}$ and $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}^{(1,0,3)}$ (Figs. B.20 and B.22) and are shown in Figs. B.23 and B.24 respectively. The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-XI}}$.

Constants: Punctured PPRF key $K_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}$, PPRF key $K_{\mathrm{SPS},E}$, Signing key SK_C, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\mathrm{SPS,IN}}$

Output: Signature $\sigma_{\mathrm{SPS,OUT}}$, or $\perp$

1. (a) Compute $r_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}, (h, \ell_{\mathrm{INP}}), (\mathrm{SK}_{\mathrm{SPS},E}, \mathrm{VK}_{\mathrm{SPS},E}, \mathrm{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},E})$.
   (b) Set $m = (v, \mathrm{ST}, w, 0)$.
   (c) If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},E}, m, \sigma_{\mathrm{SPS,IN}}) = 0$, output $\perp$.
2. (a) If $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$, compute $r_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, (h, \ell_{\mathrm{INP}}, 0))$, $(\mathrm{SK}_{\mathrm{SPS},A}, \mathrm{VK}_{\mathrm{SPS},A}, \mathrm{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},A})$.
   (b) If $(h, \ell_{\mathrm{INP}}) = (h^*, \ell^*)$, output $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}_C, m)$.
   Else, output $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}_{\mathrm{SPS},A}, m)$.

Fig. B.23. $\mathsf{Change\text{-}SPS.Prog}^{(4,3)}$

$\mathsf{Hyb}_{0,\nu-1,4\text{-XIII}}$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-XII}}$ except that while creating the ν^{th} signing key queried by $\mathcal{A}$, $\mathcal{B}$ and forms $\underline{(\mathrm{SK}_{\mathrm{SPS},C}^{(\nu,0)}, \mathrm{VK}_{\mathrm{SPS},C}^{(\nu,0)}, \mathrm{VK}_{\text{SPS-REJ},C}^{(\nu,0)}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\mathrm{SPS},C}^{(\nu,0)} = \mathcal{F}(K_{\mathrm{SPS},A}^{(\nu)}, (h^*, \ell^*, 0)))}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-XIV}}$: This experiment corresponds to $\mathsf{Hyb}_{0,\nu-1,4,0'}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}\vartheta)}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between 1/2 and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,4\text{-}\vartheta}$, for $\vartheta \in \{\mathrm{I}, \ldots, \mathrm{XIV}\}$. From the description of the hybrid experiments it follows that $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4)}(\lambda) \equiv$

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda$, Punctured PPRF keys $K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}$, Verification key VK_C, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or (Header Position POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3. (a) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   Else, set $\text{VK}_{\text{SPS},A} = \text{VK}_C$.
   (b) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \wedge [(t > t^*) \vee \underline{(t \leq 1)} \vee (h \neq h^*) \vee (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4. (a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$B$'$]$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$A$'$] \wedge [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [t \leq 1]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5. (a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6. (a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   If $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, 1)] \wedge [m_{\text{IN}} = m_{\ell^*,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
   Else if $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, 1)] \wedge [m_{\text{IN}} \neq m_{\ell^*,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, for some integer τ', set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. B.24. $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}^{(1,0,4)}$

$\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-I})}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,0')}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-XIV})}(\lambda)$. Hence, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,0')}(\lambda)| \leq \sum_{\vartheta=\text{II}}^{\text{XIV}} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}(\vartheta-\text{I}))}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}\vartheta)}(\lambda)|. \tag{B.7}$$

Claims B.34–B.46 below will show that the RHS of Eq. (B.7) is negligible and thus Lemma B.8 follows.

Claim B.34. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}I)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}II)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim B.34 uses a similar kind of logic as that employed in the proof of Claim B.24. We omit the details here. □

Claim B.35. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}II)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}III)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim B.35 resembles that of Claim B.25 with some suitable changes. The details are omitted. □

Claim B.36. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}III)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}IV)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Claim B.36 can be proven using an analogous logic as that used in the proof of Claim B.26. We omit the details here. □

Claim B.37. *Assuming* SPS *is a splittable signature scheme satisfying '*$\text{VK}_{\text{SPS-ONE}}$ *indistinguishability', for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}IV)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}V)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.37 is similar to that of Claim B.27 and, therefore, we do not provide the details here. □

Claim B.38. *Assuming* SPS *is a splittable signature scheme satisfying '*$\text{VK}_{\text{SPS-ABO}}$ *indistinguishability', for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}V)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VI)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.38 proceeds along a similar path to that of Claim B.28. We omit the details here. □

Claim B.39. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VI)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.39 employs the same type of logic as that applied in Claim B.29 and hence we do not provide the details in this case as well. □

Claim B.40. *Assuming* ACC *is a positional accumulator satisfying the 'indistinguishability of read setup' property, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VII)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VIII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-VII})}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-VIII})}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the indistinguishability of read setup property of the positional accumulator ACC using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge signing attribute string $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{U}_{\text{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ computes $r_{\text{SIG}}^* = \mathcal{F}(K, (h^*, \ell^*))$ followed by $(\widehat{\text{SK}}_{\text{SIG}}^*, \widehat{\text{VK}}_{\text{SIG}}^*) = \mathsf{SIG.Setup}(1^\lambda;\ r_{\text{SIG}}^*)$.
  4. $\mathcal{B}$ returns the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K]))$ to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{KEY}}]$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,4\text{-VII}}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Then, it creates the punctured PPRF keys $K_{\text{SPS},A}^{(\nu)}\{(h^*, \ell^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},A}^{(\nu)}, (h^*, \ell^*, 0))$ and $K_{\text{SPS},B}^{(\nu)}\{(h^*, \ell^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},B}^{(\nu)}, (h^*, \ell^*, 0))$.
  3. Next, $\mathcal{B}$ sends $n_{\text{ACC-BLK}} = 2^\lambda$, the sequence of symbol-index pairs $((x_0^*, 0), \ldots, (x_{\ell^*-1}^*, \ell^* - 1))$, and the index $i^* = 0$ to its ACC read setup indistinguishability challenger $\mathcal{C}$ and receives back $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$, where either $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ or $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_{\ell^*-1}^*, \ell^* - 1)), i^* = 0)$.
  4. After that, it generates $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  5. Then, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0, 0)$. For $j = 1, \ldots, \ell^*$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$
     - $w_j = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{j-1}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1, x_{j-1}^*)$
     - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}, 0))$

     It sets $m_{\ell^*,0}^{(\nu)} = (v_{\ell^*}^{(\nu)}, q_0^{(\nu)}, w_{\ell^*}, 0)$.

6. After that, it forms $(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},C}, \mathrm{VK}^{(\nu,0)}_{\mathrm{SPS},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-REJ},C}), (\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},D}, \mathrm{VK}^{(\nu,0)}_{\mathrm{SPS},D}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-REJ},D}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$ and generates $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \mathrm{SK}^{(\nu,0)}_{\text{SPS-ABO},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},C}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},C}, m^{(\nu)}_{\ell^*,0})$, $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},D}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},D}, \mathrm{SK}^{(\nu,0)}_{\text{SPS-ABO},D}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},D}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},D}, m^{(\nu)}_{\ell^*,0})$.
7. It gives $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w_0, v^{(\nu)}_0, K^{(\nu)}_{\mathrm{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K^{(\nu)}_{\mathrm{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K^{(\nu)}_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},E}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},C}, \\ \quad \mathrm{SK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1,0,2)}_{\mathrm{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, \\ \quad K^{(\nu)}_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]) \end{array}\right).$$

- For $\theta \in [\hat{q}_{\mathrm{SIGN}}]$, in reply to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma^{(\theta)}_{\mathrm{SIG}} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma^{(\theta)}_{\mathrm{ABS}} = (\widehat{\mathrm{VK}}^*_{\mathrm{SIG}}, \sigma^{(\theta)}_{\mathrm{SIG}})$.
- Finally, $\mathcal{A}$ output a signature σ^*_{ABS} on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its ACC read setup indistinguishability experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\mathrm{ABS}}) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its ACC read setup indistinguishability experiment.

Note that if $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,4\text{-VII}}$. On the other hand, if $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x^*_0, 0), \ldots, (x^*_{\ell^*-1}, \ell^* - 1)), i^* = 0)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$. This completes the proof of Claim B.40. □

Claim B.41. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$ and ACC is a positional accumulator satisfying the 'read enforcing' property, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,4\text{-}VIII)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}IX)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$ and $\mathsf{Hyb}_{0,\nu-1,4\text{-IX}}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} signing key provided to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,4\text{-IX}}$ it includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Constrained\text{-}Key.Prog}^{(1,0,2)}_{\mathrm{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]$ (Fig. B.21),
- $P_1 = \mathsf{Constrained\text{-}Key.Prog}^{(1,0,3)}_{\mathrm{ABS}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]$ (Fig. B.22).

We will argue that the programs P_0 and P_1 are functionally equivalent, so that, by the security of $\mathcal{IO}$ Claim B.41 follows. Clearly, the only inputs for which the outputs of the two programs might differ are those corresponding to $(h, \ell_{\mathrm{INP}}, t) = (h^*, \ell^*, 1)$. For inputs corresponding to $(h^*, \ell^*, 1)$, P_1 is programmed to output $\perp$ in case $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}$ but $\alpha =$'A', whereas, P_0 has no such condition in its programming. Now, observe that for inputs corresponding to $(h^*, \ell^*, 1)$, both the programs will assign the value 'A' to α if and only if $\mathsf{SPS.Verify}(\mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1$, where $\mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}$ is generated by running $\mathsf{SPS.Split}(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},C}, m^{(\nu)}_{\ell^*,0})$. Hence, by the correctness [Properties (i), (iii) and (v)] of the splittable signature scheme SPS, described in Definition 2.6, it is immediate that for inputs corresponding to $(h^*, \ell^*, 1)$, both programs will set $\alpha =$'A' only if $m_{\mathrm{IN}} = m^{(\nu)}_{\ell^*,0}$. Now, $m_{\mathrm{IN}} = m^{(\nu)}_{\ell^*,0}$ means $\mathrm{ST}_{\mathrm{IN}} = q^{(\nu)}_0, w_{\mathrm{IN}} = w^{(\nu)}_{\ell^*}$, and $\mathrm{POS}_{\mathrm{IN}} = 0$. Further, recall that in both the hybrid experiments $\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$ and $\mathsf{Hyb}_{0,\nu-1,4\text{-IX}}$, $(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w^{(\nu)}_0, \mathrm{STORE}^{(\nu)}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x^*_0, 0), \ldots, (x^*_{\ell^*-1}, \ell^* - 1)), i^* = 0)$. Therefore, by the read enforcing property of ACC it follows that if $w_{\mathrm{IN}} = w^{(\nu)}_{\ell^*}$ and $\mathrm{POS}_{\mathrm{IN}} = 0$, then $\mathsf{ACC.Verify\text{-}Read}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}, \pi_{\mathrm{ACC}}) = 1$ implies $\mathrm{SYM}_{\mathrm{IN}} = x^*_0$. Hence, for both the programs,

for inputs corresponding to $(h^*, \ell^*, 1)$, $\alpha =$'A' implies $\text{ST}_{\text{IN}} = q_0^{(\nu)}$ and $\text{SYM}_{\text{IN}} = x_0^*$, which in turn implies $\text{ST}_{\text{OUT}} \neq q_{\text{AC}}$. Hence, the two programs have identical outputs for inputs corresponding to $(h^*, \ell^*, 1)$ as well. Thus, the two programs are functionally equivalent. □

Claim B.42. *Assuming* ACC *is a positional accumulator satisfying the 'indistinguishability of read setup' property, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}IX)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}X)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.42 is similar to that of Claim B.40 with some appropriate modifications that are easy to figure out. □

Claim B.43. *Assuming* SPS *is a splittable signature scheme satisfying 'splitting indistinguishability', for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}X)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}XI)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.43 proceeds along a similar path as that of the proof of Claim B.30. We omit the details here. □

Claim B.44. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}XI)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}XII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.44 employs a similar type of logic as that utilized in the proof of Claim B.31. We omit the details in this case as well. □

Claim B.45. *Assuming* $\mathcal{F}$ *is a secure puncturable pseudorandom function, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}XII)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}XIII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.45 takes an analogous path as that taken by the proof of Claim B.25. The details are omitted. □

Claim B.46. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* P/poly *and* $\mathcal{F}$ *satisfies the correctness under puncturing property, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}XIII)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}XIV)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim B.46 applies the same kind of logic as that employed in the proof of Claim B.24. The details are again omitted. □

□

Lemma B.9. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* P/poly*,* ACC *is a secure positional accumulator, and* ITR *is a secure cryptographic iterator, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,(\gamma-1)')}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,\gamma)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Lemma B.9 follows an analogous path to that of Lemma B.4 of [10] with certain appropriate modifications that are easy to identify. □

Lemma B.10. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* P/poly*,* $\mathcal{F}$ *is a secure puncturable pseudorandom function,* ACC *is a positional accumulator possessing the 'indistinguishability of read setup' as well as 'read enforcing' properties, and* SPS *is a splittable signature scheme satisfying '*$\text{VK}_{\text{SPS-ONE}}$ *indistinguishability', '*$\text{VK}_{\text{SPS-ABO}}$ *indistinguishability', as well as 'splitting indistinguishability' properties, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,\gamma)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,\gamma')}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Lemma B.10 is similar to that of Lemma B.3 of [10] with some appropriate readily identifiable changes. □

Lemma B.11. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* P/poly *and* ACC *is a positional accumulator having 'indistinguishability of read setup' and 'read enforcing' properties, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,(t^{*(\nu)}-1)')}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,5)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Lemma B.11 is similar to that of Lemma B.5 of [10] with some appropriate changes that are easy to determine. □

Lemma B.12. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$*, $\mathcal{F}$ is a secure puncturable pseudorandom function,* SPS *is a splittable signature scheme possessing the '*$\mathrm{VK}_{\mathrm{SPS\text{-}REJ}}$ *indistinguishability' property, and* PRG *is a secure injective pseudorandom generator, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,6)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Lemma B.12 is similar to that of Lemma B.6 of [10]. □