Attribute-Based Signatures for Unbounded Languages from Standard Assumptions

Yusuke Sakai[1], Shuichi Katsumata[1,2], Nuttapong Attrapadung[1], and Goichiro Hanaoka[1]

[1] AIST, Japan
[2] The University of Tokyo, Japan

Abstract. Attribute-based signature (ABS) schemes are advanced signature schemes that simultaneously provide fine-grained authentication while protecting privacy of the signer. Previously known expressive ABS schemes support either the class of deterministic finite automata and circuits from standard assumptions or Turing machines from the existence of indistinguishability obfuscations.
In this paper, we propose the first ABS scheme for a very general policy class, all deterministic *Turing machines*, from a standard assumption, namely, the Symmetric External Diffie-Hellman (SXDH) assumption. We also propose the first ABS scheme that allows *nondeterministic finite automata* (NFA) to be used as policies. Although the expressiveness of NFAs are more restricted than Turing machines, this is the first scheme that supports *nondeterministic* computations as policies.
Our main idea lies in abstracting ABS constructions and presenting the concept of *history of computations*; this allows a signer to prove possession of a policy that accepts the string associated to a message in zero-knowledge while also hiding the policy, regardless of the computational model being used. With this abstraction in hand, we are able to construct ABS for Turing machines and NFAs using a surprisingly weak NIZK proof system. Essentially we only require a NIZK proof system for proving that a (normal) signature is valid. Such a NIZK proof system together with a base signature scheme are, in turn, possible from bilinear groups under the SXDH assumption, and hence so are our ABS schemes.

Keywords: attribute-based signatures, Groth-Sahai proofs, structure-preserving signatures, Turing machines, nondeterministic finite automata

1 Introduction

Attribute-based signature (ABS), initiated by Maji, Prabhakaran, and Rosulek [MPR11], is a cryptographic primitive that simultaneously allows fine-grained access control on user authentication and protection of users' privacy. In the so-called key-policy ABS[3], which is the focus of this work, each signer

[3] The other type is called signature-policy, where the roles of policies and attributes are swapped.

is associated with his/her own policy and obtains a signing key for this policy from an authority, who possesses the master key. Using the signing key, a signer can sign any message associated with any attribute subjected to the condition that the policy is satisfied by this attribute. ABS provides privacy in the sense that a signature hides the policy that is used to sign the message. That is, no information on the policy beyond the fact that it is satisfied by the associated attribute will be leaked to the verifier. ABS has many natural applications such as anonymous credential [SSN09], attribute-based messaging [MPR11], and secret leaking [MPR11].

One of the central research themes on ABS is to expand the expressiveness of policies that can be supported by the scheme. Results in this direction include the scheme by Okamoto and Takashima [OT11], which supports non-monotone span programs as policies. Tang, Li, and Liang [TLL14] proposed a scheme that supports bounded-depth circuits (albeit their scheme relies on strong tools, namely, multilinear maps). Nandi and Pandit [NP15] proposed several schemes including one that supports deterministic finite automata (DFA). One of the most expressive scheme to date is the ABS scheme proposed by Sakai, Attrapadung, and Hanaoka [SAH16]; their scheme supports unbounded-depth unbounded-size circuits and is based on bilinear maps under standard assumptions. On the other hand, recently, ABS schemes that support policies on the opposite end of the spectrum from circuits, namely Turing machines, were constructed by Datta, Dutta, and Mukhopadhyay [DDM17]. However, their scheme requires the strong assumption of the existence of indistinguishability obfuscations. Therefore, we still do not know of any ABS schemes achieving the ultimate goal of supporting Turing machines with unbounded-length inputs that rely on a well-established assumption such as a static assumption over bilinear groups. Considering the current situation that all the known encryption scheme counterparts of ABS (i.e., attribute-based encryption (ABE)) seem to require the power of indistinguishability obfuscation, one may think that it is simply out of our reach to construct ABS for Turing machines from standard assumptions.

Besides its theoretical interests, ABS for Turing machines, which naturally support unbounded languages, has a practical benefit. Here, by unbounded languages, we mean an ABS scheme where different signatures have attribute strings of different lengths. Policies are associated with signing keys, and thus the policy needs to accept variable-length attribute strings. To see the benefit of this primitive, let us suppose a company where the manager wants each employee to send an email on behalf of the company to the customers that are assigned to this employee. An ABS for unbounded languages provide a natural solution for this setting. In this solution, a manager of the company possesses a master secret key of an ABS scheme. Then the manager assigns to each employee a signing key with a policy. This policy describes what addresses the employee is allowed to submit an email. Then the employee signs an email using the destination address of a customer as an attribute string. The flexibility of ABS for Turing machines (or finite automata) is helpful in this scenario. The manager can specify the policy using a regular expression such as `*@division.example.com`, to restrict the employee to send an email to some division of a customer company. In this application, Turing machines or finite automata provide an expressive way to describe a policy. Moreover, unbounded attribute strings quite meet to this scenario, since attribute strings are set to be an email address, which has a variable length.

1.1 Our Contribution

In this paper, we present an attribute-based signature scheme over bilinear maps that allows us to use an arbitrary deterministic Turing machine as the policy from *standard assumptions*. In particular, unlike ABS schemes for policies such as non-uniform circuits, our scheme allows one to associate an *unbounded*-length string as an attribute to the message. Due to the uniform nature of Turing machines, we depart from the conventional ABS constructions and incorporate new ideas and techniques to cope with the unboundedness of the policies. Notably, we abstract ABS constructions, and use the concept of *history of computations* to prove that the signer is in possession of a Turing machine (or in general, some computational model) that accepts the string associated to the message. Furthermore, we build on the idea which we call the *locality of rewriting* to prove the above statement in zero-knowledge. These abstraction and ideas allow us to circumvent the standard intuition that we would require a strong NIZK proof system for proving a valid computation of a Turing machine to construct ABS schemes. Our scheme is reasonably efficient compared to other cryptographic schemes that support Turing machine type computations (e.g., ABE for Turing machines).

Our scheme satisfies perfect privacy and unforgeability. The scheme is proven secure under the symmetric external Diffie-Hellman (SXDH) assumption over bilinear groups. More precisely, our scheme is based on the Groth-Sahai proof system and a structure-preserving signature, both of which can be proven secure under the SXDH assumption. The signature size is $O(T^2)$ where T is an upper bound for the running time of a Turing machine, which is specified by the signer. The size of a signing key is $O(|Q| \cdot |\Gamma|^4)$ where $|Q|$ is the number of the states of the Turing machine and $|\Gamma|$ is the size of the tape alphabet. We emphasize that in spite of its expressiveness, our scheme only requires a standard static assumption (SXDH) over bilinear groups. This could be a striking contrast to the case of attribute-based encryption (ABE) for Turing machines, where available schemes [GKP$^+$13,AS16] require much stronger tools, such as indistinguishability obfuscation.

In addition to the above main contribution, we also present another ABS scheme whose policy class is restricted to *nondeterministic finite automata* (NFA).[4] This scheme is the first scheme that supports *nondeterministic* computation as a policy. The policy classes of all the previously known ABS schemes

[4] In terms of languages that machines accept, NFA is equivalent to a subclass of Turing machines called read-only right-moving Turing machines. Both accept the class of regular languages.

are only deterministic: (non-)monotone span programs, Boolean circuits, deterministic finite automata, and deterministic Turing machines. In addition, this scheme gains efficiency compared with the main scheme. Namely, it has the signature of size $O(|w|)$ where $|w|$ is the length of the input of the finite automaton. The size of a signing key is $O(|Q|^2 \cdot |\Sigma|)$ where $|Q|$ is the number of the states of the automaton, and $|\Sigma|$ is the size of the alphabet. In particular, the dependency on the alphabet size is down to $|\Sigma|$ from $|\Gamma|^4$.

1.2 Paper Organization

We dedicate the next section (Sect. 2) for describing the intuition of our schemes. We then begin with some preliminaries and formal definitions in Sect. 3 including the formal definition of Turing machines and related notions. Our main ABS scheme for Turing machines is then presented in Sect. 4, while its security proof is given in Sect. 5. The scheme for nondeterministic finite automata is given in Sect. 6.

2 Difficulties and Our Approach

2.1 Naive Ideas and Their Limitations

As a warm up, we provide some of the obstacles when trying to construct ABS schemes for Turing machines from previously known tools and techniques. One of the most naive approaches may be to base the construction on ABS schemes supporting circuits [SAH16], since theoretically, circuits are already quite powerful. However, this idea would not work because of the differences between the models of computations; circuits are non-uniform but Turing machines are uniform. Specifically, there is simply no easy way to embed a uniform computational model (i.e., Turing machine) into the signing key starting from an ABS scheme that only supports non-uniform computational models (i.e. circuits) as the key-policy. Another approach may be to base the scheme on expressive attribute-based encryption (ABE) schemes, however, the problem of this approach is that, at least in general, ABE schemes do not provide the anonymity property, which is an essential requirement for ABS schemes. Furthermore, no ABE schemes for Turing machines based on standard assumptions are known. One may also consider starting from policy-based signatures [BF14] or functional signatures [BGI14]. However, this does not provide a successful solution as well, since all the known policy-based signature schemes only support pairing-product equations for describing a policy, or if we want to support any NP relations, we need to employ general zero-knowledge. The same holds for functional signatures, namely, it supports any NP relations, at the cost of general zero-knowledge.

In a high level, all of the above obstacles seem to boil down to the problem of not having any efficient NIZK proof for proving correct computation of a Turing machine. As an example, ABS for circuits were achievable [SAH16], since Groth-Sahai proof systems can be used to prove correct circuit computation in zero-knowledge. Indeed, if we had an efficient NIZK proof for such an unbounded language, we could have taken another route and convert an ABE scheme for Turing machines into its ABS scheme counterpart by proving that the secret key associated with the Turing machine satisfies.

2.2 Our Approach

We now explain the technical overview of our ABS scheme for Turing machines and show how we circumvent the above problem. In particular, our construction does not rely on any strong NIZK proof system; essentially it only requires a NIZK proof system for proving that a signature is valid. In the following, we assume some familiarity on the standard notion of Turing machines and finite automata, which will be explained in Sect. 3 and Sect. 6.1, respectively. We first provide a high level approach to constructing ABS schemes using the concept of *history of computations*. Then, for simplicity, we explain how to use our idea for the case where the policies are "deterministic" finite automata, and provide an overview on how to further extend it to Turing machines via an idea we call the *locality of rewriting*.

Abstract Approach: Using a History of Computations. We first step back and give a high level background on how previous ABS schemes were constructed. One of the most standard ways of constructing an ABS scheme is the "certificate" method [MPR11,SAH16]. In this method, the authority issues each signer a digital signature signed on the user's policy, which will serve as a certificate for the user's signing privilege. When the signer decides to sign a message with some attribute, the signer proves in zero-knowledge that he possesses a valid signature on some policy in addition that the attribute satisfies that certain policy. Therefore, in theory, we can always use general NIZKs for NP languages to construct ABS schemes at the cost of a very inefficient scheme. In light of this, much of the efforts for constructing ABS schemes are centered around constructing an efficient NIZK proof system for proving that a policy embedded in the signature satisfies the attribute while also hiding the policy. In the following, we explain the abstract approach we take for constructing such NIZK proof systems.

As we have mentioned above, the central difficulty in constructing an ABS scheme for complex (unbounded) computational models, such as finite automata or Turing machines stems from the fact that we do not have sufficiently expressive and efficient NIZK proof systems. Toward this end, we take the approach of expressing a computation in a set of sufficiently simple formulae that can be handled by a simple and efficient NIZK proof system. In particular, our key idea is to use a *history of computations* to prove that the hidden policy satisfies the attribute. Here, by a history of computations, we mean a sequence s_{init}, s_1, ..., s_n of "snapshots" of a machine[5], which expresses how the computation proceeded in a step-by-step manner. For simplicity, for the time being, let us assume that

[5] Here, one can think of the "snapshot" as the state the algorithm is in. For example, a snapshot of a Turing machine is whatever written in the working tape, the state

the policy, i.e., the machine, is public. Then, the main advantage of the above approach is that even though it may be hard to express the actual computation of the policy into a simple formulae, once given a history of computations s_{init}, $s_1, \ldots, s_n$ and the policy, it may be much easier to express an algorithm that validates this sequence into a simple formulae. Note that this abstract idea can be used for any type of computational models (bounded or unbounded) as long as one can appropriately define the history of computations while being able to express them into simple formulae. For example, as we show later, it may be much easier to prove that, (s_{i-1}, s_i) for all i follow the (public) transition function of a machine, rather than writing out the automaton as a very large and complex formulae.

Now, taking into consideration that the policies must also be hidden, the following depicts our key idea on the whole.

$$s_{\text{init}} \longrightarrow \boxed{s_1} \longrightarrow \boxed{s_2} \longrightarrow \cdots \longrightarrow \boxed{s_n} \in \boxed{F}$$

Here, consider the input w to the machine M to be implicitly included in the public state s_{init}. The gray box indicates that the snapshots s_i must be hidden and F indicates the set of accepting states of the machine M, which in some cases must be hidden since they may leak information on M. If the above can be proven in zero-knowledge, then a verifier would be convinced that the signer is in possession of a machine M which accepts the input w associated to the message. Now, we can break the problem of proving possession of a valid history of computations into three sub-problems: (i) prove that each hidden snapshots s_i are valid snapshots, (ii) prove that each transition of the snapshots s_i to s_{i+1} is consistent with the signer's policy and (iii) prove that the final snapshot s_n is in the accepting states F. Finally, to use this idea of a history of computations to construct an ABS scheme, we must make sure that the above policy and accepting states F showing up in the sub-problems (ii) and (iii), respectively, are certified by the authority. As one may think, the most difficult part of the sub-problems will turn out to be item (ii). In the following, we first provide a detailed explanation on how to use the above idea to construct an ABS scheme for deterministic automata. We then build on that idea to provide an explanation for the more complex Turing machine.

History of Computations for Finite Automata. Before getting into details, we define a finite automaton. Informally, a finite automaton M is defined by a set of states Q, a transition function $\delta \colon Q \times \Sigma \to Q$ and a set of accepting states $F \subseteq Q$, where Σ denotes the input alphabet. Next, we define what a history of computations is for the case of finite automata. Since a history of computations is simply a sequence of snapshots of the finite automaton given an attribute string $w = w_1 \cdots w_n$, we can express this simply as a sequence of states q_{init}, $q_1, \ldots,$ q_n; the computation starts with the automaton being at the initial state q_{init}, then moving to state q_1 after reading w_1 (i.e., $q_1 \leftarrow \delta(q_{\text{init}}, w_1)$), then moving to

it is in, and the position of the head. Furthermore, we use the term machine loosely to express some computational model such as Turing machines or automata.

state q_2 after reading $w_2, \dots,$ and finally reading w_n and moving to q_n, which is an accepting state. Here, note that the states Q and the accepting states F are different for each automaton, and hence must be hidden. In particular, we use the following as the history of computations for finite automata:

$$\langle q_{\text{init}}, w_1, \boxed{q_1} \rangle \longrightarrow \langle \boxed{q_1}, w_2, \boxed{q_2} \rangle \longrightarrow \cdots \longrightarrow \langle \boxed{q_{n-1}}, w_n, \boxed{q_n} \rangle, \ \boxed{q_n} \in \boxed{F}, \quad (1)$$

where once again the gray box indicates that they are to be hidden. Informally, the previous snapshot s_i now corresponds to $\langle q_i, w_{i+1}, q_{i+1} \rangle$.

Now that we have defined what the history of computations is, we must show how to solve the aforementioned problems: Below we look at first how to prove that each snapshot $\langle q_i, w_{i+1}, q_{i+1} \rangle$ are valid while hiding the automaton being used and how the authority certifies the automaton $M = (Q, \delta, F)$ to a signer. A naive approach would be to encode the transition function δ to a single large input-output table of size $|Q| \times |\Sigma|$ where the entries of the table is of the form $\langle q, w, q' \rangle$, sign this large table, and use this signature as a certificate for the signer. Then, the signer can prove sequentially in zero-knowledge that each snapshot is included in the table by using a NIZK proof system that supports simple vector-matrix multiplications.[6] However, this table-based approach cannot be secure because the table is variable-length. Namely, since each automaton may have different numbers of states $|Q|$, the size of the tables varies with the automata. Therefore if we use this variable-length table as a witness for the NIZK proof system, the length of the proof may also vary. Hence the signature (i.e., the zero-knowledge proof) leaks information on the automaton. We emphasize that the anonymity notion for ABS schemes requires that even when two automata have different numbers of the states, signatures produced by the two different automata as policies should be indistinguishable from each other, provided that these two automata accept the same string.

Instead, we let the authority issue the signer a signing key as the set of signatures on each entry of the table as follows:

$$\left\{ \theta_{q,w} = \mathsf{Sign}(\mathsf{sk}, \langle q, w, \delta(q,w) \rangle) \right\}_{(q,w) \in Q \times \Sigma}, \quad \left\{ \bar{\theta}_{\bar{q}} = \mathsf{Sign}(\mathsf{sk}, \bar{q}) \right\}_{\bar{q} \in F}.$$

Here the number of signatures is roughly $|Q| \times |\Sigma|$. In particular, since Q and Γ are polynomial sizes in the security parameter, the total number of signatures is polynomial. Now, proving knowledge of a history of computations becomes much simpler; the signer picks the respective signatures $\theta_{q_i, w_{i+1}}$ from the set of signatures $\{\theta_{q,w}\}$ and proves that they are valid signatures in zero-knowledge and proves that the final state q_n opens to some signature in $\{\bar{\theta}_{\bar{q}}\}$. The signer also shows that each transition of the snapshots are consistent with the signer's automaton by using the same commitments for each state $q \in Q$ and proving in zero knowledge that the committed signatures are valid. Since the number of steps it takes for an automaton to terminate is the same as the length of the

[6] In particular, it proves that the table includes an entry of the form $\langle q, w, q' \rangle$ while hiding which entry it is. This can be accomplished by viewing the table as a matrix and using unit vectors to indicate the row to pick up.

input string, the aforementioned problem concerning the variable proof length is resolved. Finally, a subtle technical detail is that in the actual construction, the authority includes some nonce in the signatures $\theta_{q,w}$ and $\bar{\theta}_{\bar{q}}$ so that they are tied to a unique automaton to prevent collusion attacks.

An informal intuition on the security is as follows: let us consider for a moment a situation where a malicious signer wants to generate an attribute-based signature even though the automaton assigned to him does not accept the attribute string $w = w_1 \cdots w_n$. In this case, since the automaton does not accept the attribute string, we have that any sequence $q_{\mathrm{init}}, q_1, \ldots, q_n$ leading to an accepting state must deviate from δ. Specifically, at least one adjacent pair (q_i, q_{i+1}) must satisfy $\delta(q_i, w_{i+1}) \neq q_{i+1}$. Since the signer is never issued a signature on this triple $\langle q_i, w_{i+1}, q_{i+1}\rangle$, he will not be able to execute the proof of knowledge.

Extending the Idea to Turing Machines. We now explain our history approach for the case of Turing machines. Again, the goal is to let the authority certify a signer's transition function in such a way that the signer can efficiently prove knowledge of a history of computations.

Firstly we briefly recapitulate the notion of Turing machines. A Turing machine is specified by a set of state Q, a transition function $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\mathtt{left}, \mathtt{stay}, \mathtt{right}\}$, an initial state $q_{\mathrm{init}} \in Q$, and an accepting state q_{acc}, where Γ is the tape alphabet. In the following, we assume the initial state q_{init} and the accepting state q_{acc} are set to be special symbols that are common to all Turing machines. Specifically, the set of accepting states F can be made public. A Turing machine starts its computation with an input w on its working tape, the head at the leftmost cell. Then the machine moves the head left and right while rewriting the cells of the working tape one by one. We say the Turing machine accepts input w if the Turing machine reaches the accept state q_{acc}.

A snapshot of a Turing machine can be identified by specifying (1) the state, (2) the contents of the working tape, and (3) the position of the head. We encode this information by a string $uqv \in \Gamma^* \times Q \times \Gamma^*$ where u, $v \in \Gamma^*$ and $q \in Q$. This encoding specifies that the state is q, the contents of the working tape are uv, and the head is pointing at the leftmost symbol of v. For example, we encode a snapshot in which (1) the machine takes the state $q \in Q$, (2) the head is on the fourth symbol, and (3) the tape contents are $w_1w_2w_3w_4w_5w_6 \in \Gamma^*$ by the encoding

$$s = w_1w_2w_3 \; q \; w_4w_5w_6.$$

Using this encoding, one way to define the history of computations of a Turing machine is as follows:

$$\begin{aligned} &q_{\mathrm{init}} \; w_1w_2w_3w_4w_5w_6 \longrightarrow \boxed{w_1' \; q_1 \; w_2w_3w_4w_5w_6} \\ &\qquad \longrightarrow \boxed{w_1'w_2' \; q_2 \; w_3w_4w_5w_6} \longrightarrow \cdots \longrightarrow q_{\mathrm{acc}} \; \boxed{w_1''w_2''w_3''w_4''w_5''w_6''} \in F. \end{aligned}$$

With this history of computations in hand, we now must resolve the aforementioned three sub-problems (i), (ii), and (iii), of which the most difficult part is problem (ii), where we have to prove that adjacent snapshots $s_{i-1} \longrightarrow s_i$ are valid transitions. In the case of finite automata, the solution was to sign on all possible pairs of the form $\langle q, w, \delta(q, w)\rangle$, which specifies that the transition from q to $\delta(q, w)$ is valid. Unfortunately, the simple approach of signing on all possible valid pairs of snapshots $\langle s, t\rangle$ will not work for Turing machines. Due to the unboundedness of the model of computation, the length of a working tape is unbounded, and hence there are an unbounded, or even infinite, number of possible valid pairs of snapshots which the authority must sign.

"Locality of rewriting" is our key insight to overcome this difficulty. To explain this, let us consider a snapshot $s = abcdeqxfg$, i.e., the current state being q, the content written on the tape being $abcdexfg$, and the head pointing to x. For simplicity, in the following argument, we always use a, b, c, d, e, f, g to denote arbitrary symbols in $\mathit{\Gamma}$ and use x to denote the symbol which the Turing machine reads next. Then, if the transition function satisfies $\delta(q, x) = (q', x', \texttt{left})$, the next snapshot would be $t = abcdq'ex'fg$, where q' is the next state and x' is the symbol written in place of x. Observe that the symbol x' and its position in t is determined by the two neighbors of the corresponding positions in s, namely, q and x, as the transition function directs the machine to rewrite x with x' and moves the head left. Similarly, the symbol b and its position in state t is determined by the three neighbors a, c, d in s. Namely, since none of its three neighbors are pointed by the head in s, the symbol b is unchanged. In general, any symbol in a succeeding snapshot is determined by the four neighbors in the current snapshot: the symbol in the same position, its left symbol, and the two symbols on its right. Fig. 1 illustrates all the cases of the four neighbors determining the symbols in the succeeding snapshot, in the case that the head moves to left.[7] In this figure, the upper tapes denote the preceding snapshot, while the lower tapes denote its succeeding snapshot. The grayed boxes in the upper tapes denote the neighbors that determine the grayed box in the lower line. Although we included both cases 1 and 2 for completeness, the grayed boxes hold the same meaning since all the four neighbors in the preceding snapshot are constituted only from tape symbols. In particular, there are 5 cases depending on the position of the state q is in the four preceding neighbors.

Using this locality of rewriting, the authority signs all the possible occurring patterns of the above grey boxes, which consists of the four neighbors in the preceding snapshot and the one symbol in the succeeding snapshot. In more detail, in the case that $\delta(q, x) = (q', x', \texttt{left})$, the authority signs on the following five types of tuples:

$$\begin{array}{llllll}
(a, & b, & c, & d, & b &), \\
(c, & d, & e, & q, & d &), \\
(d, & e, & q, & x, & q' &), \\
(e, & q, & x, & f, & e &), \\
(q, & x, & f, & g, & x' &)
\end{array} \qquad (2)$$

[7] Similar illustrations can be obtained for the case that the head stays and moves to the right with the same idea.

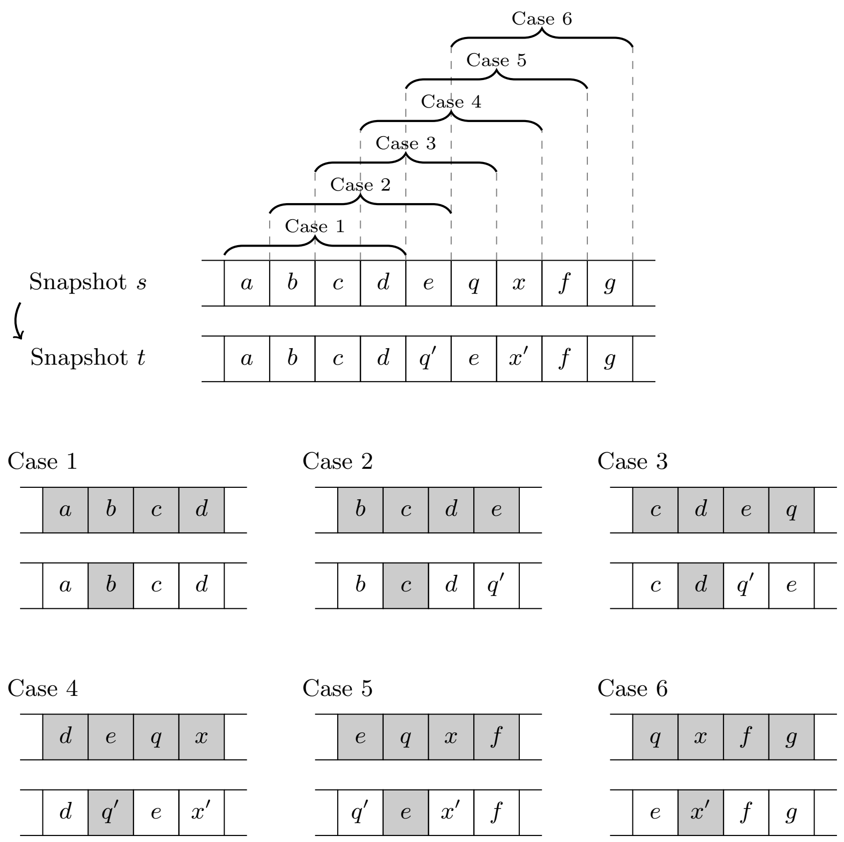

Fig. 1. All the patterns of "local" changes when a snapshot s becomes a snapshot t via a transition $\delta(q, x) = (q', x', \texttt{left})$.

for all possible choices of a, b, c, d, e, f, and $g \in \Gamma$.[8] Finally, the authority creates signatures for all possible choices of $(q, x) \in Q \times \Gamma$ and provides the set of these signatures to the signer as the signing key. Here, we did not need to consider the tuple (b, c, d, e, c), since this is captured by the first tuple of Eq. (2).

With this set of signatures, the signer can prove knowledge of a history of computations

$$s_{\text{init}} \longrightarrow \boxed{s_1} \longrightarrow \boxed{s_2} \longrightarrow \cdots \longrightarrow \boxed{s_n}$$

in the following way. For each adjacent snapshots $s_{i-1} = u_1 u_2 \cdots u_n$ and $s_i = v_1 v_2 \cdots v_n$ such that

$$\boxed{u_1 u_2 \cdots u_n} \longrightarrow \boxed{v_1 v_2 \cdots v_n},$$

[8] The other two cases `right` and `stay` are done similarly.

the signer proves knowledge of the possession of signatures for the following tuples:

$$\begin{array}{lllllll}
\langle\ \text{\textvisiblespace}, & u_1, & u_2, & u_3, & v_1 & & \rangle, \\
\langle\ u_1, & u_2, & u_3, & u_4, & v_2 & & \rangle, \\
\langle\ u_2, & u_3, & u_4, & u_5, & v_3 & & \rangle, \\
 & & \vdots & & & & , \\
\langle\ u_{n-3}, & u_{n-2}, & u_{n-1}, & u_n, & v_{n-2} & & \rangle, \\
\langle\ u_{n-2}, & u_{n-1}, & u_n, & \text{\textvisiblespace}, & v_{n-1} & & \rangle, \\
\langle\ u_{n-1}, & u_n, & \text{\textvisiblespace}, & \text{\textvisiblespace}, & v_n, & & \rangle.
\end{array}$$

where $\text{\textvisiblespace}$ denotes the blank symbol included in Γ. This can be done if the transition $s_{i-1} \longrightarrow s_i$ follows the transition function, since the signer can pick the signature on each tuple from the signatures signed on the tuples in Eq. (2). This way, the signer can prove the consistency of transitions, namely, *both* the fact that the cells pointed at by the head are rewrote following the transition function and the fact that the cells not pointed at by the head are untouched. The blank symbols on the left and right sides of $u_1 u_2 \cdots u_n$ are included in order to also treat the corner cased where the header points to the leftmost or two rightmost cells. For a more formal discussion, we refer the readers to the AttrSign algorithm in Sect. 4.

An intuition on the security is similar to the case of finite automata. Let us suppose that a malicious signer whose Turing machine does not accept an attribute string tries to produce the above zero-knowledge proof. In this case, for any choice of a history of computations, at least one adjacent snapshots $s_{i-1} = u_1 \cdots u_n$ and $s_i = v_1 \cdots v_n$ deviate from the transition function. Informally, this would imply that, at least for some v_i, the tuple $\langle u_{i-1}, u_i, u_{i+1}, u_{i+2}, v_i\rangle$ does not match any of the tuples in Eq. (2). Therefore, the malicious signer is not issued a signature on such a tuple, hence he would not be able to execute the zero-knowledge proof. The actual proof will be more contrived since we add nonces in multiple places to prevent mix-and-match attacks.

Applying the Idea to Nondeterministic Finite Automata. Our approach can be extended to support nondeterministic finite automata (NFA). Before we explain how to do so, we first state some difficulty of obtaining ABS for NFA. A naive idea of converting an NFA into a *deterministic* finite automaton (DFA) and then using ABS for DFA (e.g.,[NP15]) would not work since the equivalent DFA would suffer an exponential blowup in the number of states (see, for example, [Sip96]). Again we cannot rely on expressive zero-knowledge proofs that can express nondeterministic computation, because such proofs would be quite expensive, if possible at all.

Instead, we proceed with our idea of a history of computations. In our history approach, the authority issues a signature for each transition $q \longrightarrow \delta(q, w)$. Recall that in a nondeterministic finite automaton, there are multiple choices of transitions and the actual choice will be chosen nondeterministically. Let us consider a signer whose automaton has two choices of transition q' and q'' when the automaton is in state q and reading w. Nondeterminism means that if at least one choice of q' or q'' leads to an accepting state, the automaton accepts the input. Then, if the signer has two signatures on both $\langle q, w, q' \rangle$ and $\langle q, w, q'' \rangle$, then the signer can build a history of computations, together with a signature as in Eq. (1), by choosing either a signature on $\langle q, w, q' \rangle$ or a signature on $\langle q, w, q'' \rangle$ depending on which choice of the transition leads to an accepting state. Based on this idea, our scheme for nondeterministic finite automata lets the authority issue signatures on $\langle q, w, q' \rangle$ and $\langle q, w, q'' \rangle$ for both q' or q''. This way, to sign with a nondeterministic finite automata, the signer firstly computes a history of computations nondeterministically, then picks the signatures that correspond to the nondeterministic transition, and proves the knowledge of such signatures.

The size of signing key is $O(|Q|^2 \cdot |\Sigma|)$, where $|Q|$ is the number of the states of the automaton, and $|\Sigma|$ is the size of the alphabet. The factor $|Q| \cdot |\Sigma|$ corresponds to the fact that there are $|Q| \cdot |\Sigma|$ entries for the transition function, and the other factor $|Q|$ corresponds to the fact that there are at most $|Q|$ nondeterministic choices for each entry.

3 Preliminaries

We say that a function $f\colon \mathbb{N} \to \mathbb{R}$ is negligible if for any $c \in \mathbb{N}$ there exists x_0 satisfying that for any $x \leq x_0$ it holds that $f(x) \leq 1/x^c$. We denote by ϵ the empty string. For a string u we denote by $|u|$ the length of u. For two strings u and v we denote by uv the concatenation of u and v. For a set S, we denote by $\mathcal{P}(S)$ the powerset of S.

Turing Machines. We give the definition of Turing machines. We consider deterministic Turing machines in this paper, but will often omit the word 'deterministic'.

Definition 1. *A Turing machine over the input alphabet* $\{0, 1\}$ *is a tuple* $(Q, \Gamma, \delta, q_{\text{init}}, q_{\text{acc}}, q_{\text{rej}})$ *where*

- Q *is a finite set of the states,*
- Γ *is a finite set of the tape alphabet which contains symbols* 0 *and* 1*, a left endmarker* \$*, and a blank symbol* ␣*.*
- $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\texttt{left}, \texttt{stay}, \texttt{right}\}$ *is a transition function,*
- $q_{\text{init}} \in Q$ *is the initial state,*
- $q_{\text{acc}} \in Q$ *is the accept state, and*
- $q_{\text{rej}} \in Q$ *is the reject state,* $q_{\text{rej}} \neq q_{\text{acc}}$*.*

We require that the states and the tape alphabet do not intersect, namely, $Q \cap \Gamma = \emptyset$. We also require that all Turing machines do not rewrite the left endmarker \$ with another symbol and do not move the head beyond the endmarker. Formally, we require that the transition function satisfies $\delta(q, \$) = (q', \$, \texttt{right})$

or $\delta(q, \$) = (q', \$, \mathtt{stay})$ for any q and some q'. Furthermore, we require that once the machine reaches to either the accept state q_{acc} or the reject state q_{rej}, the machine never moves to another state, namely, $\delta(q_{\mathrm{acc}}, x) = (q_{\mathrm{acc}}, x, \mathtt{stay})$ and $\delta(q_{\mathrm{rej}}, x) = (q_{\mathrm{rej}}, x, \mathtt{stay})$ for any $x \in \mathit{\Gamma}$.

We define the notion of a configuration. A configuration describes the entire snapshot of the machine and the tape, including the state $q \in Q$, the contents of the tape, and the position of the head. Formally, we say that a string $t \in \mathit{\Gamma}^* \times Q \times \mathit{\Gamma}^*$ is a configuration of a Turing machine. In a configuration t, the string obtained by removing the unique $q \in Q$ describes the contents of the tape. The occurrence of q simultaneously describes the state of the machine and the position of the head. The occurrence q specifies q as the current state of the machine, and also specifies the symbol at the right of q as the position of the head.

We then formally define the notion of transition.

Definition 2. *Let $M = (Q, \mathit{\Gamma}, \delta, q_{\mathrm{init}}, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ be a Turing machine and $s = uzqxv$ be a configuration for it, where u, $v \in \mathit{\Gamma}^*$, $z \in \mathit{\Gamma} \cup \{\epsilon\}$, and $q \in Q$. We denote $s \underset{M}{\longrightarrow} t$ if one of the followings holds.*

1. *$\delta(q, x) = (q', x', \mathtt{left})$ and $t = uq'zx'v$,*
2. *$\delta(q, x) = (q', x', \mathtt{stay})$ and $t = uzq'x'v$,*
3. *$\delta(q, x) = (q', x', \mathtt{right})$, and $t = uzx'q'v$.*

We denote by $\underset{M}{\longrightarrow}^*$ the reflexive transitive closure of $\underset{M}{\longrightarrow}$. We say that a Turing machine $M = (Q, \mathit{\Gamma}, \delta, q_{\mathrm{init}}, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ accepts $w \in \{0, 1\}^*$ if

$$q_{\mathrm{init}}\$w\text{␣}\cdots\text{␣} \underset{M}{\longrightarrow}^* q_{\mathrm{acc}}v$$

for some $v \in \mathit{\Gamma}^*$ and sufficiently long ␣ $\cdots$ ␣.[9] Here we assume that all Turing machines halt after moving its head to the leftmost symbol.

During the computation of a Turing machine M, if the current configuration t is $uqxv$ where u, $v \in \mathit{\Gamma}^*$, $x \in \mathit{\Gamma}$, and $q \in Q$, we say that the machine *reads* the symbol x, or the symbol x is *being read*. Furthermore, if the machine makes a transition $uqv \underset{M}{\longrightarrow} u'q'v'$, where u, v, u', $v' \in \mathit{\Gamma}^*$ and q, $q' \in Q$, then we say that the machine *moves* from the state q to the state q'.

Attribute-Based Signatures. We then define the syntax of attribute-based signatures for Turing machines. A formal treatment on Turing machines can be found in Section 3. An attribute-based signature scheme for Turing machines consists of the following four algorithms.

$\mathsf{AttrSetup}(1^k) \to (\mathsf{pp}, \mathsf{msk})$**.** The setup algorithm takes as an input a security parameter 1^k and outputs a public parameter pp and a master secret key msk.

[9] Since our definition of transition does not automatically expand the tape with blank symbols, we need to explicitly pad with blank symbols. The length can be bounded by the number of the steps until the machine halts.

$\mathsf{AttrGen}(\mathsf{pp}, \mathsf{msk}, M) \rightarrow \mathsf{sk}$**.** The key generation algorithm takes as inputs a public parameter pp, a master secret key msk, and a description of a Turing machine M and outputs a signing key sk.

$\mathsf{AttrSign}(\mathsf{pp}, \mathsf{sk}, T, w, m) \rightarrow \sigma$**.** The signing algorithm takes as inputs a public parameter pp, a signing key sk, an upper bound T for the running time of the Turing machine, a string $w \in \{0,1\}^*$, and a message $m \in \{0,1\}^*$ and outputs a signature σ.

$\mathsf{AttrVerify}(\mathsf{pp}, T, w, m, \sigma) \rightarrow 1/0$**.** The verification algorithm takes as inputs a public parameter pp, an upper bound T for the running time, a string w, and a message m and outputs a bit 1 or 0.

As the correctness condition, we require that for any $k \in \mathbb{N}$, any $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{AttrSetup}(1^k)$, any Turing machine M, any $\mathsf{sk} \leftarrow \mathsf{AttrGen}(\mathsf{pp}, \mathsf{msk}, M)$, any $T \in \mathbb{N}$, any string $w \in \{0,1\}^*$ which is accepted by M within T steps, any message $m \in \{0,1\}^*$, and any $\sigma \leftarrow \mathsf{AttrSign}(\mathsf{pp}, \mathsf{sk}, T, w, m)$ it holds that $\mathsf{AttrVerify}(\mathsf{pp}, T, w, m, \sigma) = 1$.

We then define two security requirements for attribute-based signature schemes. The first requirement is anonymity, which requires that no information on the Turing machine used to generate a signature does leak. The other requirement is unforgeability. It requires that no collusion can generate a signature under attributes, if their policies do not accept the attributes.

Definition 3. *An attribute-based signature scheme is perfectly anonymous, if for any $k \in \mathbb{N}$, any $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{AttrSetup}(1^k)$, any Turing machines M_0 and M_1, any $\mathsf{sk}_0 \leftarrow \mathsf{AttrGen}(\mathsf{pp}, \mathsf{msk}, M_0)$ and $\mathsf{sk}_1 \leftarrow \mathsf{AttrGen}(\mathsf{pp}, \mathsf{msk}, M_1)$, any $T \in \mathbb{N}$, any string $w \in \{0,1\}^*$ which is accepted within T steps by both M_0 and M_1, and any message $m \in \{0,1\}^*$, the distributions $\mathsf{AttrSign}(\mathsf{pp}, \mathsf{sk}_0, T, w, m)$ and $\mathsf{AttrSign}(\mathsf{pp}, \mathsf{sk}_1, T, w, m)$ are identical, where the probability is taken over the randomness of the* $\mathsf{AttrSign}$ *algorithm.*

Definition 4. *An attribute-based signature scheme is unforgeable if for any probabilistic polynomial-time adversary $\mathcal{A}$ the probability that the adversary $\mathcal{A}$ wins in the following game against a challenger is negligible in k.*

1. *The challenger generates a public parameter* pp *and a master secret key* msk *by running $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{AttrSetup}(1^k)$. The challenger maintains a set of pairs of the form (M, sk) where M is a Turing machine and* sk *is a signing key. It is initially set to $\emptyset$. The challenger sends* pp *to the adversary $\mathcal{A}$.*
2. *Then the adversary $\mathcal{A}$ is allowed to issue key generation queries and signing queries.*
   - *For a key generation query M, the challenger searches for a tuple (M, sk) in the set. If it is not found, the challenger generates a signing key* sk *by running $\mathsf{sk} \leftarrow \mathsf{AttrGen}(\mathsf{pp}, \mathsf{msk}, M)$ and stores (M, sk) in the set. The challenger returns* sk *to the adversary $\mathcal{A}$.*
   - *For a signing query (M, T, w, m), the challenger verifies that M terminates within T steps on input w. If not, it returns $\perp$. Then the challenger searches for a tuple (M, sk) in the set. If it is not found, the challenger*

generates a signing key sk *by running* $\mathsf{sk} \leftarrow \mathsf{AttrGen}(\mathsf{pp}, \mathsf{msk}, M)$ *and stores* (M, sk) *in the set. The challenger generates a signature by running* $\sigma \leftarrow \mathsf{AttrSign}(\mathsf{pp}, \mathsf{sk}, T, w, m)$ *and returns* σ *to the adversary* $\mathcal{A}$.

3. *The adversary* $\mathcal{A}$ *outputs* $(T^*, w^*, m^*, \sigma^*)$ *and halts.*
4. *The adversary* $\mathcal{A}$ *wins the game if the following conditions hold: (i)* $\mathsf{AttrVerify}(\mathsf{pp}, T^*, w^*, m^*, \sigma^*) = 1$*; (ii) the adversary* $\mathcal{A}$ *did not issue any key generation* M *which accepts* w^* *within time* T^**; (iii) the adversary* $\mathcal{A}$ *did not issue a signing query* (M, T^*, w^*, m^*) *for any* M.

Bilinear Groups. A bilinear group parameter generation algorithm $\mathcal{G}$ takes a security parameter 1^k as an input and outputs a bilinear group parameter $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g})$ where p is a prime, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are order-p multiplicative groups, $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map, and g and $\tilde{g}$ is generators of $\mathbb{G}_1$ and $\mathbb{G}_2$. Our concrete scheme is based on the symmetric external Diffie-Hellman (SXDH) assumption. We omit the formal definitions of this assumption due to the page limitation.

Groth-Sahai Proofs. The Groth-Sahai proof system [GS12] is a non-interactive proof system which can prove the satisfiability of algebraic equations over bilinear groups called pairing product equations. A pairing product equation has a form of

$$\prod_{i=1}^{n} e(\mathcal{A}_i, \mathcal{Y}_i) \prod_{i=1}^{m} e(\mathcal{X}_j, \mathcal{B}_j) \prod_{i=1}^{n} \prod_{j=1}^{m} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{i,j}} = T,$$

where $\mathcal{A}_i \in \mathbb{G}_1$, $\mathcal{B}_j \in \mathbb{G}_2$, $\gamma_{i,j} \in \mathbb{Z}_p$, and $T \in \mathbb{G}_T$ are public constants, and $\mathcal{X}_j \in \mathbb{G}_1$ and $\mathcal{Y}_i \in \mathbb{G}_2$ are secret assignments.

The Groth-Sahai proof system consists of the five algorithms $\mathsf{WISetup}$, $\mathsf{WIProve}$, $\mathsf{WIVerify}$, $\mathsf{ExtSetup}$, and $\mathsf{Extract}$. The common reference string generation algorithm $\mathsf{WISetup}$ takes as an input a bilinear group parameter gk and outputs a common reference string crs. The proof algorithm $\mathsf{WIProve}$ takes as inputs a common reference string crs, a statement (public constants) x, and a witness (secret assignments) w, and outputs a proof π. The verification algorithm $\mathsf{WIVerify}$ takes as inputs a common reference string crs, a statement x, and a proof π and outputs a bit 1 or 0. The extractable common reference string generation algorithm takes as an input a bilinear group parameter gk and outputs an extractable common reference string crs and an extraction key ek. The extraction algorithm $\mathsf{Extract}$ takes as inputs an extractable common reference string crs, an extraction key ek, a statement x, and a proof π and outputs a witness w. As the correctness condition, we require that for any $k \in \mathbb{N}$, any $\mathsf{gk} \leftarrow \mathcal{G}(1^k)$, any $\mathsf{crs} \leftarrow \mathsf{WISetup}(\mathsf{gk})$ any statement x and its witness w, it holds that $\mathsf{WIVerify}(\mathsf{crs}, x, \mathsf{WIProve}(\mathsf{crs}, x, w)) = 1$. The formal definitions of the security of proof systems are postponed to the full version.

The Groth-Sahai proof system proves the satisfiability of the pairing product equations in the following way. Firstly, the proof algorithm generates commitments to each element of the satisfying assignment. Secondly, it generates proof components which prove the satisfying assignments behind the commitments surely satisfies the statement in question. We call these commitments generated in the proving process Groth-Sahai commitments. The Groth-Sahai proof system can be instantiated from the SXDH assumption. In this case, a Groth-Sahai commitment is constituted by two source group elements for each witness element, and a proof component is constituted by eight group elements for each pairing product equation. See [GS12] for further details.

Structure-Preserving Signatures. A structure-preserving signature scheme consists of the three algorithms Kg, Sign, and Verify. The key generation algorithm takes as an input a bilinear group parameter gk and a message-length 1^n and outputs a verification key vk and a signing key sk. The signing algorithm Sign takes as inputs a signing key sk and a message $m \in {\mathbb{G}_1}^n$ and outputs a signature θ. The verification algorithm Verify takes as inputs a verification key vk, a message m, and a signature θ and outputs a bit 1 or 0. As the correctness condition, we require that for any $k \in \mathbb{N}$, any $n \in \mathbb{N}$, any $\mathsf{gk} \leftarrow \mathcal{G}(1^k)$, any $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(\mathsf{gk})$, and any $m \in {\mathbb{G}_1}^n$, it holds that $\mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$. In addition, we require that the verification algorithm consist of the set of pairing-product equations, which enables us to prove the knowledge of signatures on either public or secret messages. Such a structure-preserving signature scheme can be instantiated from the SXDH assumption [KPW15]. The formal definition of unforgeability and an instantiation of a structure-preserving signature scheme from the SXDH assumption is postponed to the full version.

Collision-Resistant Hash Functions. A hash function family consists of the two algorithms $\mathcal{H}$ and Hash. The hashing key generation algorithm $\mathcal{H}$ takes as an input a security parameter 1^k and outputs a hashing key hk. The deterministic hashing algorithm Hash takes as inputs a hashing key hk and a message $m \in \{0,1\}^*$ and outputs a hash value h. We say that a hash function family is collision resistant if for any probabilistic polynomial-time algorithm $\mathcal{A}$ the probability $\Pr[\mathsf{hk} \leftarrow \mathcal{H}(1^k); (m, m') \leftarrow \mathcal{A}(\mathsf{hk}) : \mathsf{Hash}(\mathsf{hk}, m) = \mathsf{Hash}(\mathsf{hk}, m')]$ is negligible in k. We assume that the length of a hash value h is determined only from a security parameter k. We denote by ℓ this length of a hash value.

4 Attribute-Based Signatures for Turing Machines

Notations. In our scheme, given a Turing machine that describes a policy, we need to modify this Turing machine slightly in order to resist chosen-message attacks. The input of the modified Turing machine consists of two parts. The first is the hash value of the message to be signed, while the other is an actual input to the original (unmodified) Turing machine. The modified Turing machine ignores the first part and accepts the input if and only if the original Turing machine accepts the latter part of the input.

The formal definition is as follows. For a Turing machine M, we denote by $\overline{M}$ another Turing machine obtained by the following modifications to M. Let

$h = h_1 \cdots h_\ell \in \{0,1\}^\ell$ be a string (a hash value) of length ℓ. Let w be the original input to M.

- $\overline{M}$ begins with the configuration $q_{\text{init}}\$hw$.
- It skips the input h. More precisely, in the notation of configurations, it runs the computations in the following sequence:

$$q_{\text{init}}\$h_1 \cdots h_\ell w \underset{\overline{M}}{\longrightarrow} \$q_1 h_1 \cdots h_\ell w \underset{\overline{M}}{\longrightarrow} \cdots \underset{\overline{M}}{\longrightarrow} \$h_1 h_2 \cdots h_{\ell-1} q_\ell h_\ell w.$$

- It rewrites h_ℓ with \$. More formally,

$$\$h_1 h_2 \cdots h_{\ell-1} q_\ell h_\ell w \underset{\overline{M}}{\longrightarrow} \$h_1 h_2 \cdots h_{\ell-1} q_{\ell+1} \$w$$

  where $q_{\ell+1}$ serves as a simulated M's initial state.
- It starts the simulation of M using the new \$ as M's endmarker and w as M's input.
- If the simulated M accepts w, $\overline{M}$ moves the head to its endmarker \$ as

$$q_{\text{acc}}\$h_1 \cdots h_{\ell-1}\$u.$$

  for some u and accepts hw. Otherwise $\overline{M}$ rejects hw.

Construction. In the construction of our scheme, we assume that the tape alphabet is a subset of $\{0, 1, \ldots, (p-1)/2\}$, and the set of states is a subset of $\{-1, -2, \ldots, -(p-1)/2\}$, where p is the order of the underlying bilinear groups. Furthermore, we assume that the initial state q_{init}, the accept state q_{acc}, and the reject state q_{rej} is respectively equal to -1, -2, and -3. For brevity, we denote a group element $[a]_1 \in \mathbb{G}_1$ as $[a]$. The description of our scheme is as follows.

$\mathsf{AttrSetup}(1^k)$**.** Given the security parameter 1^k, generate a bilinear group parameter $\mathsf{gk} \leftarrow \mathcal{G}(1^k)$, a witness-indistinguishable common reference string for Groth-Sahai proofs $\mathsf{crs} \leftarrow \mathsf{WISetup}(\mathsf{gk})$, two key pairs of the structure-preserving signature scheme $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(\mathsf{gk}, 1^6)$, and a hashing key $\mathsf{hk} \leftarrow \mathcal{H}(1^k)$. Set $\mathsf{pp} \leftarrow (\mathsf{gk}, \mathsf{crs}, \mathsf{vk}, \mathsf{hk})$ and $\mathsf{msk} \leftarrow \mathsf{sk}$, and output $(\mathsf{pp}, \mathsf{msk})$.

$\mathsf{AttrGen}(\mathsf{pp}, \mathsf{msk}, M)$**.** Given a public parameter pp, a master secret key msk, and a description M of a Turing machine, let $\overline{M} = (Q, \Gamma, \delta, q_{\text{init}}, q_{\text{acc}}, q_{\text{rej}})$ be the modified Turing machine as defined above. Choose a random integer $\tau \leftarrow \mathbb{Z}_p$.[10] Then generate the following set of signatures.[11]
- For all a, b, c, and $d \in \Gamma$, generate

$$\theta[\![a, b, c, d]\!] \leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau], [a], [b], [c], [d], [b])). \quad (3)$$

[10] This extra component of the messages is for resisting collusion attacks. Without this component, colluding signers may produce a forgery of the attribute-based signature scheme by mixing their certificates and forming a history of transitions that is not allowed to each of the signers.

[11] See Fig. 1 and Eq. (2) for intuition. The five tuples in Eq. (2) are implemented in Eqs. (3)–(7) for the case that the head moves to left. The other two cases are implemented similarly.

– For all c, d, $e \in \Gamma$, and $q \in Q$, generate

$$\theta[\![c,d,e,q]\!] \leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[c],[d],[e],[q],[d])). \tag{4}$$

– For all $q \in Q$ and x, d, e, f, and $g \in \Gamma$, generate the following signatures.
  - If $\delta(q,x) = (q', x', \texttt{left})$, generate

$$\begin{aligned}\theta[\![d,e,q,x]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[d],[e],[q],[x],[q'])), & (5)\\ \theta[\![e,q,x,f]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[e],[q],[x],[f],[e])), & (6)\\ \theta[\![q,x,f,g]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[q],[x],[f],[g],[x'])). & (7)\end{aligned}$$

  - If $\delta(q,x) = (q', x', \texttt{stay})$, generate

$$\begin{aligned}\theta[\![d,e,q,x]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[d],[e],[q],[x],[e])),\\ \theta[\![e,q,x,f]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[e],[q],[x],[f],[q']),\\ \theta[\![q,x,f,g]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[q],[x],[f],[g],[x']).\end{aligned}$$

  - If $\delta(q,x) = (q', x', \texttt{right})$, generate

$$\begin{aligned}\theta[\![d,e,q,x]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[d],[e],[q],[x],[e]),\\ \theta[\![e,q,x,f]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[e],[q],[x],[f],[x']),\\ \theta[\![q,x,f,g]\!] &\leftarrow \mathsf{Sign}(\mathsf{sk}, ([\tau],[q],[x],[f],[g],[q']).\end{aligned}$$

Output the signing key sk as

$$\begin{aligned}\mathsf{sk} = (M, \tau, &(\theta[\![a,b,c,d]\!])_{(a,b,c,d)\in\Gamma\times\Gamma\times\Gamma\times\Gamma},\\ &(\theta[\![c,d,e,q]\!])_{(c,d,e,q)\in\Gamma\times\Gamma\times\Gamma\times Q},\\ &(\theta[\![d,e,q,x]\!])_{(d,e,q,x)\in\Gamma\times\Gamma\times Q\times\Gamma},\\ &(\theta[\![e,q,x,f]\!])_{(e,q,x,f)\in\Gamma\times Q\times\Gamma\times\Gamma},\\ &(\theta[\![q,x,f,g]\!])_{(q,x,f,g)\in Q\times\Gamma\times\Gamma\times\Gamma}).\end{aligned}$$

$\mathsf{AttrSign}(\mathsf{pp},\mathsf{sk},T,w,m)$**.** Let $\tilde{T}$ be the upper bound of $\overline{M}$'s required steps for simulating M's computation up to T steps.

1. Compute $h \leftarrow \mathsf{Hash}(hk, \langle T, w, m\rangle)$.
2. Let $t_0 = q_{\mathrm{init}}\$hw(\llcorner)^{\tilde{T}-|h|-|w|}$. Compute the sequence of configurations of $\overline{M}$ as
$$t_0 \xrightarrow[\overline{M}]{} t_1 \xrightarrow[\overline{M}]{} \cdots \xrightarrow[\overline{M}]{} t_{\tilde{T}}$$
that reaches to the accept state q_{acc}, where $t_{\tilde{T}} = q_{\mathrm{acc}}u$ for some u. Let $t_i = t_{i,0}\cdots t_{i,\tilde{T}+1}$. Let $t_{i,-1} = t_{i,\tilde{T}+2} = t_{i,\tilde{T}+3} = \llcorner$ for each $i \in \{0,\ldots,\tilde{T}\}$.
3. Let
$$\theta_{i,j} \leftarrow \theta[\![t_{i,j-1}, t_{i,j}, t_{i,j+1}, t_{i,j+2}]\!]$$
for each $i \in \{0,\ldots,\tilde{T}-1\}$ and each $j \in \{0,\ldots,\tilde{T}+1\}$.

4. Compute
   - a Groth-Sahai commitment $\psi^{(\tau)}$ to $[\tau]$,
   - Groth-Sahai commitments $\psi^{(t)}_{i,j}$ to $[t_{i,j}]$ for all[12] $i \in \{1, \dots, \tilde{T}\}$ and $j \in \{0, \dots, \tilde{T}+1\}$ except for $t_{\tilde{T},0}$, which is equal to q_{acc} and thus treated as a public constant, and
   - Groth-Sahai commitments $\psi^{(\theta)}_{i,j}$ to $\theta_{i,j}$ for all[13] $i \in \{0, \dots, \tilde{T}-1\}$ and $j \in \{0, \dots, \tilde{T}+1\}$.
5. Generate a Groth-Sahai proof $\pi_{i,j}$ for the equation

$$\mathsf{Verify}(\mathsf{vk}, ([\tau], [t_{i,j-1}], [t_{i,j}], [t_{i,j+1}], [t_{i,j+2}], [t_{i+1,j}]), \theta_{i,j}) = 1$$

   for each $i \in \{0, \dots, \tilde{T}-1\}$ and $j \in \{0, \dots, \tilde{T}+1\}$.
6. Output the signature σ as

$$\sigma = \Bigg(\psi^{(\tau)}, (\psi^{(t)}_{i,j})_{(i,j)\in\{1,\dots,\tilde{T}\}\times\{0,\dots,\tilde{T}+1\}\setminus\{(\tilde{T},0)\}}, \\ (\psi^{(\theta)}_{i,j})_{(i,j)\in\{0,\dots,\tilde{T}-1\}\times\{0,\dots,\tilde{T}+1\}}, \\ (\pi_{i,j})_{(i,j)\in\{0,\dots,\tilde{T}-1\}\times\{0,\dots,\tilde{T}+1\}}\Bigg).$$

$\mathsf{AttrVerify}(\mathsf{pp}, T, w, m, \sigma)$**.** Compute $h \leftarrow \mathsf{Hash}(\mathsf{hk}, \langle T, w, m\rangle)$ and verify all the proofs in σ under this T, w, m, and h. Here to verify that the initial configuration is valid and that the last state is q_{acc}, the initial configuration $t_0 = q_{\mathrm{init}}\$hw(\llcorner\!\lrcorner)^{\tilde{T}-|h|-|w|}$ and the state q_{acc} of the last configuration are treated as public constants in the proofs. If all the proofs are verified as valid, output 1. Otherwise output 0.

5 Security of Our Scheme

In this section, we provide the security proof of our main scheme.

Before the proof of the main theorem, we introduce a notion of an authorized pair of configurations and present a lemma related to authorized pairs.

Definition 5. *Let $M = (Q, \Gamma, \delta, q_{\mathrm{init}}, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ be a Turing machine. Let $s = s_0 \cdots s_{\tilde{T}+1}$ and $t = t_0 \cdots t_{\tilde{T}+1}$ be strings of alphabets $Q \cup \Gamma$ such that $s \in \Gamma^* \times Q \times \Gamma^*$. Let $s_{-1} = s_{\tilde{T}+2} = s_{\tilde{T}+3} = t_{-1} = t_{\tilde{T}+2} = t_{\tilde{T}+3} = \llcorner\!\lrcorner$. Let q be the state of s and let x be the symbols that the head is reading in s. We say that the pair (s, t) is authorized at position j with respect to M if*

[12] We do not need $\psi^{(t)}_{i,j}$ for $i = 0$, since these cases correspond to the initial configuration, which is public.

[13] We do not need $\psi^{(\theta)}_{i,j}$ for $i = \tilde{T}$, since these commitments are used to bind a configuration t_i and the next configuration t_{i+1}, but $t_{\tilde{T}}$ is the last configuration.

– *It holds that* $\delta(q,x) = (q', x', \texttt{left})$ *and* $(s_{j-1}, s_j, s_{j+1}, s_{j+2}, t_j)$ *is equal to either of the following:*

$$(a,b,c,d,b),\ (c,d,e,q,d),\ (d,e,q,x,q'),\ (e,q,x,f,e) \text{ or } (q,x,f,g,x')$$

for some a, b, c, d, e, f, *and* $g \in \Gamma$.

– *It holds that* $\delta(q,x) = (q', x', \texttt{stay})$ *and* $(s_{j-1}, s_j, s_{j+1}, s_{j+2}, t_j)$ *is equal to either of the following:*

$$(a,b,c,d,b),\ (c,d,e,q,d),\ (d,e,q,x,e),\ (e,q,x,f,q'), \text{ or } (q,x,f,g,x')$$

for some a, b, c, d, e, f, *and* $g \in \Gamma$.

– *It holds that* $\delta(q,x) = (q', x', \texttt{right})$ *and* $(s_{j-1}, s_j, s_{j+1}, s_{j+2}, t_j)$ *is equal to either of the following:*

$$(a,b,c,d,b),\ (c,d,e,q,d),\ (d,e,q,x,e),\ (e,q,x,f,x'), \text{ or } (q,x,f,g,q')$$

for some a, b, c, d, e, f, *and* $g \in \Gamma$.

When (s,t) *is authorized at position* j *with respect to* M *due to one of the tuples of the above forms, we also say that the symbol* t_j *is authorized by that tuple.*

The following lemma plays an important role during the security proof. At a high level, it allows us to argue about valid transitions by only considering authorized configuration pairs.

Lemma 1. *Let* M *be a Turing machine and let* $s = s_0 \cdots s_{\tilde{T}+1}$ *and* $t = t_0 \cdots t_{\tilde{T}+1}$ *be strings in* $(Q \cup \Gamma)^*$. *If* $s \underset{M}{\not\longrightarrow} t$ *and* $s \in \Gamma^* \times Q \times \Gamma^*$, *then there is a position* j *such that* (s,t) *is not authorized at position* j *with respect to* M.

Proof. We prove the contraposition. Namely, assuming (s,t) is authorized at all positions $j \in \{0, \ldots, \tilde{T}+1\}$ with respect to M, we will prove that either $s \underset{M}{\not\longrightarrow} t$ or $s \notin \Gamma^* \times Q \times \Gamma^*$ holds. Toward this end we assume that (s,t) is authorized at all positions $j \in \{0, \ldots, \tilde{T}+1\}$ with respect to M and that $s \in \Gamma^* \times Q \times \Gamma^*$ and will prove that $s \underset{M}{\longrightarrow} t$.

Let us set

$$s = uzqxv$$

where $u \in \Gamma^*$, $z \in \Gamma \cup \{\epsilon\}$, $q \in Q$, $x \in \Gamma$, and $v \in \Gamma^*$. Without loss of generality we assume that if $u = \epsilon$, then $z = \epsilon$. Since (s,t) is authorized by M, we have that $|s| = |t|$. Thus we can set

$$t = u'\zeta\eta\chi v'$$

where $|u'| = |u|$, $|\zeta| = |z|$, $|\eta| = 1$, $|\chi| = 1$, $|v'| = |v|$. Similarly, we assume that if $\zeta = \epsilon$, then $u' = \epsilon$.

– Suppose $\delta(q,x) = (q', x', \texttt{left})$. Firstly it does not occur that $(u,z) = (\epsilon, \epsilon)$, since in this case the head does not move left. Hence we have that $z \neq \epsilon$.

- All the symbols in u' are authorized by either (a, b, c, d, b) or (c, d, e, q, d), and hence $u' = u$.
- The symbol ζ is authorized by (d, e, q, x, q'), and hence $\zeta = q'$.
- The symbol η is authorized by (e, q, x, f, e), and hence $\eta = z$.
- The symbol χ is authorized by (q, x, f, g, x'), and hence $\chi = x'$.
- All the symbols in v' are authorized by (a, b, c, d, b), and hence $v' = v$.

Therefore we have that $t = uq'zx'v$, and thus $s \xrightarrow[M]{} t$.

– Suppose $\delta(q, x) = (q', x', \texttt{stay})$.

- All symbols in u' are authorized by either (a, b, c, d, b) or (c, d, e, q, d), and hence $u' = u$.
- The symbol ζ is authorized by (d, e, q, x, e), and hence $\zeta = z$.
- The symbol η is authorized by (e, q, x, f, q'), and hence $\eta = q'$.
- The symbol χ is authorized by (q, x, f, g, x'), and hence $\chi = x'$.
- All the symbols in v' are authorized by (a, b, c, d, b), and hence $v' = v$.

Therefore we have that $t = uzq'x'v$, and thus $s \xrightarrow[M]{} t$.

– Suppose $\delta(q, x) = (q', x', \texttt{right})$.

- All symbols in u' are authorized by either (a, b, c, d, b) or (c, d, e, q, d), and hence $u' = u$.
- The symbol ζ is authorized by (d, e, q, x, e), and hence $\zeta = z$.
- The symbol η is authorized by (e, q, x, f, x'), and hence $\eta = x'$.
- The symbol χ is authorized by (q, x', f, g, q'), and hence $\chi = q'$.
- All the symbols v' are authorized by (a, b, c, d, b), and hence $v' = v$.

Therefore we have that $t = uzx'q'v$, and thus $s \xrightarrow[M]{} t$.

Hence, since we have $s \xrightarrow[M]{} t$ for all cases, the lemma holds. □

The main theorem is as follows.

Theorem 1. *Assuming the Groth-Sahai proof system is perfectly witness indistinguishable and computationally extractable, the structure-preserving signature scheme is existentially unforgeable, and the hash function family is collision resistant, the attribute-based signature scheme is perfectly anonymous and unforgeable.*

Instantiating all the primitives from the SXDH assumption, we have the following as a corollary.

Corollary 1. *If the SXDH assumption holds for $\mathcal{G}$, our instantiation is perfectly anonymous and unforgeable.*

We then prove the main theorem.

Theorem 2. *Assuming the Groth-Sahai proof system is perfectly witness indistinguishable, the attribute-based signature scheme is perfectly anonymous.*

Proof. The theorem immediately follows from the witness indistinguishability of the proof system. Fix two Turing machines M_0 and M_1 that accept a string $w \in \{0,1\}^*$ within T steps. Since both machines accept the same string, and both $\overline{M_0}$ and $\overline{M_1}$ halt with the same time $\tilde{T}$, we have two sequences of configurations

$$t_0 \xrightarrow[\overline{M_0}]{} t_1 \xrightarrow[\overline{M_0}]{} \cdots \xrightarrow[\overline{M_0}]{} t_{\tilde{T}}$$

for M_0 and

$$s_0 \xrightarrow[\overline{M_1}]{} s_1 \xrightarrow[\overline{M_1}]{} \cdots \xrightarrow[\overline{M_1}]{} s_{\tilde{T}}$$

for M_1, in which $t_0 = s_0$. Since the public constants in the proofs, in particular the initial configuration and the accept state q_{acc}, are determined by t_0 for M_0 and s_0 for M_1, the two proofs share the same public constants. Therefore, thanks to the witness indistinguishability of the proof system, both proofs are equally distributed. □

Theorem 3. *Assuming the Groth-Sahai proof system is perfectly witness indistinguishable and computationally extractable, the structure-preserving signature scheme is existentially unforgeable, and the hash function family is collision resistant, the attribute-based signature scheme is unforgeable.*

Proof. For a given hash value $h \in \{0,1\}^\ell$, we define a Turing machine $M[h]$ as follows: It compares the first ℓ symbols of the input with the hardwired hash value h; if they are identical, it moves its head to the endmarker and accepts the input, otherwise rejects the input.

Let us consider the following sequence of games.

Game 0. This is identical to the game in the definition of unforgeability.

Game 1. In the response to each signing query (M, T, w, m), the challenger uses the Turing machine $M[h]$ where $h \leftarrow \mathsf{Hash}(\mathsf{hk}, \langle T, w, m\rangle)$ instead of M. Namely, every time (M, T, w, m) is queried, the challenger generates a signing key for $M[h]$ and use this signing key to generate a signature to be returned to the adversary. The signing key for $M[h]$ will be generated every time a query is issued, and will not be reused.

Game 2. In this game, we add the following additional clause to the winning condition: *(iv) the adversary $\mathcal{A}$ did not issue any signing query* (M, T, w, m) *that satisfies* $\mathsf{Hash}(\mathsf{hk}, \langle T, w, m\rangle) = \mathsf{Hash}(\mathsf{hk}, \langle T^*, w^*, m^*\rangle)$.

Game 3. In the response to either key generation queries or signing queries, the random integer $\tau \leftarrow \mathbb{Z}_p$ is equal to any of the responses to the previous queries, the challenger returns $\perp$.

Game 4. In this game the challenger replaces the common reference string crs in the public parameter pp with the extractable one $\mathsf{crs} \leftarrow \mathsf{ExtSetup}(\mathsf{gk})$.

Let us denote by W_i the event that the winning conditions are satisfied in Game i. From the triangle inequality, we have that

$$\Pr[W_0] = \sum_{i=1}^{4}(\Pr[W_{i-1}] - \Pr[W_i]) + \Pr[W_4] \leq \sum_{i=1}^{4}|\Pr[W_{i-1}] - \Pr[W_i]| + \Pr[W_4]. \tag{8}$$

To complete the proof, we then need to bound each term in this inequality.

Lemma 2. *Assuming the witness indistinguishability of the Groth-Sahai proof system,* $|\Pr[W_0] - \Pr[W_1]| = 0$.

Proof (of Lemma 2). Observe that in both games the challenger proves the same set of equalities regardless of which Turing machine is used to generate a signing key. Therefore, due to the perfect witness indistinguishability of the proof system, the challenger's responses are equally distributed. Thus the lemma holds. □

Lemma 3. *Assuming the collision resistance of the hash function family, we have that* $|\Pr[W_1] - \Pr[W_2]|$ *is negligible.*

Proof (of Lemma 3). Let F_2 be the event that the winning conditions (i), (ii), and (iii) are satisfied but the condition (iv) is not satisfied. From the difference lemma we have that $|\Pr[W_1] - \Pr[W_2]| \leq \Pr[F_2]$. To bound this probability, we construct an algorithm which attacks the collision resistance of the hash function family. The construction is as follows: The algorithm takes as input a hashing key hk; using this hashing key, the algorithm sets up the rest of the components of pp and sends it to $\mathcal{A}$; the algorithm keeps the signing key of the structure-preserving signature scheme; key generation queries and signing queries are dealt with as in the description of the games using the signing key of the structure-preserving signature scheme; when the adversary halts with an output $(T^*, w^*, m^*, \sigma^*)$, the algorithm searches for a signing query (T, w, m) that satisfies $\mathsf{Hash}(\mathsf{hk}, \langle T, w, m\rangle) = \mathsf{Hash}(\mathsf{hk}, \langle T^*, w^*, m^*\rangle)$; if a query is found, the algorithms outputs $(\langle T, w, m\rangle, \langle T^*, w^*, m^*\rangle)$ as a collision. Let us argue that whenever F_2 occurs the algorithm breaks the collision resistance of the hash function family. Since we have that the winning condition (iv) is not met, the algorithm successfully finds a query (T, w, m) that satisfies $\mathsf{Hash}(\mathsf{hk}, \langle T, w, m\rangle) = \mathsf{Hash}(\mathsf{hk}, \langle T^*, w^*, m^*\rangle)$. Since we also have that the winning condition (iii) is met, we have that $\langle T, w, m\rangle \neq \langle T^*, w^*, m^*\rangle$. Therefore, we have that whenever F_2 occurs, the algorithm successfully breaks the collision resistance of the hash function family. This implies that the probability $\Pr[F_2]$ is negligible. □

Lemma 4. *The quantity* $|\Pr[W_2] - \Pr[W_3]|$ *is negligible.*

Proof (of Lemma 4). Let F_3 be the event that any of the integers τ generated in either key generation queries or signing queries is equal to any of the integers generated in the previous queries. From the difference lemma we have that $|\Pr[W_2] - \Pr[W_3]| \leq \Pr[F_3]$. Let $F_{3,i}$ be the event that the integer in the i-th (key generation or signing) query is equal to any of the integers in the previous queries. Hence $F_3 = F_{3,1} \vee \cdots \vee F_{3,q}$ where q is the total of the numbers of the key generation and signing queries. Then we have that $\Pr[F_3] \leq \sum_{i=1}^{q} \Pr[F_{3,i}] = \sum_{i=1}^{p} \frac{i-1}{p} = \frac{q(q-1)}{2p}$, which is negligible. □

Lemma 5. *Assuming the computational extractability of the Groth-Sahai proof system, we have that* $|\Pr[W_3] - \Pr[W_4]|$ *is negligible.*

Proof (of Lemma 5). Given an adversary $\mathcal{A}$ that plays either Game 3 or Game 4, we can construct an algorithm $\mathcal{B}$ that distinguishes a witness-indistinguishable common reference string from an extractable one. The construction of $\mathcal{B}$ is as follows: $\mathcal{B}$ is given a common reference string crs, and then it sets up all the other components of the public parameter pp; $\mathcal{B}$ runs the adversary $\mathcal{A}$ with its input pp; when $\mathcal{A}$ issues a key generation or a signing query, it responds as described in the games using the signing key of the structure-preserving signature, which was generated by $\mathcal{B}$ itself; when $\mathcal{A}$ halts, $\mathcal{B}$ outputs a bit 1 if the winning conditions are satisfied, otherwise outputs 0. Since the simulation of Game 3 and Game 4 is perfect, and then by the extractability of the Groth-Sahai proof system, $|\Pr[W_3] - \Pr[W_4]|$ is negligible. □

Finally, we bound the probability $\Pr[W_4]$.

Lemma 6. *Assuming the unforgeability of the structure-preserving signature scheme, we have that* $\Pr[W_4]$ *is negligible.*

Proof (of Lemma 6). In Game 4, let us consider having the challenger extract the witness behind the forgery $(T^*, w^*, m^*, \sigma^*)$. Let us denote this witness as

$$[\tau^*], ([t^*_{i,j}])_{(i,j)\in\{1,\dots,\tilde{T}^*\}\times\{0,\dots,\tilde{T}^*+1\}\setminus\{(\tilde{T}^*,0)\}}, \\ ([\theta^*_{i,j}])_{(i,j)\in\{0,\dots,\tilde{T}^*-1\}\times\{0,\dots,\tilde{T}^*+1\}}, \quad (9)$$

where $\tilde{T}^*$ is the upper bound for the running time determined by T^*.

Given this notion, let us consider the following algorithm $\mathcal{B}$ which internally simulates Game 4 and attacks the existential unforgeability of the structure-preserving signature scheme: Given a verification key vk of the signature scheme as an input, $\mathcal{B}$ sets up the rest of the public parameter pp of the attribute-based signature scheme as in Game 4; then $\mathcal{B}$ runs $\mathcal{A}$ providing pp as $\mathcal{A}$'s input; when $\mathcal{A}$ issues a key generation query or a signing query, $\mathcal{B}$ issues signing queries to its own challenger, and using the challenger's responses to answer $\mathcal{A}$'s query as described in Game 4; once $\mathcal{A}$ halts with a forgery $(T^*, w^*, m^*, \sigma^*)$, $\mathcal{B}$ extracts an entire witness from this forgery; finally, $\mathcal{B}$ searches the set of the witness for a forgery of the structure-preserving signature scheme; if a forgery is found, $\mathcal{B}$ outputs this forgery, otherwise outputs $\perp$.

Notice that in this construction of $\mathcal{B}$, due to the computational extractability of the Groth-Sahai proof system, whenever $\mathcal{A}$ satisfies the winning condition, $\mathcal{B}$ successfully obtains a witness that satisfies the proved equations. Therefore, to complete the proof, we argue that whenever $\mathcal{B}$ successfully obtains a witness, $\mathcal{B}$ successfully outputs a forgery against the structure-preserving signature scheme. This implies that $\Pr[W_4]$ is negligible, which concludes the proof.

The argument proceeds with a case analysis. Let us consider the following conditions.

1. The extracted τ^* is equal to one of the random integer τ generated in a response to a key generation query.

2. The extracted τ^* is equal to one of the random integer τ generated in a response to a signing query.
3. The extracted τ^* is equal to none of the above two types of τ's.

These three cases are clearly comprehensive. In the last case, any of the witness $\theta^*_{i,j}$, that is a part of the extracted witness as above, serves as a valid forgery against the underlying signature scheme, since $\mathcal{B}$ only issues signing queries which do not include τ^* in the messages.

Next, we argue that in the first two cases $\mathcal{B}$ successfully outputs a forgery.

Let us set $h^* = \mathsf{Hash}(\mathsf{hk}, \langle T^*, w^*, m^* \rangle)$. Suppose the first case has occurred. Let M be the Turing machine that is used in the response to the key generation query whose random integer τ is equal to τ^*. Due to the change introduced in Game 3, there is a unique key generation query that satisfies this. Because of the winning condition (ii), we have that M does not accept w^* in time T^*, hence $\overline{M}$ does not accepts h^*w^* in time $\tilde{T}^*$ where $\tilde{T}^*$ is the upper bound of the running time of $\overline{M}$ constructed from M. In this case, $\mathcal{B}$ issues a set of signing queries that corresponds to the transition function of $\overline{M}$, which $\mathcal{B}$ then provides as the signing key to $\mathcal{A}$. Suppose the second case has occurred. Let (M, T, w, m) be the signing query where $\mathcal{B}$ uses τ^* as the random integer to create the signature. By the change we made in Game 3, there exists at most one signing query that satisfies this. In this case, owing to the change we made in Game 1, to respond to this signing query, $\mathcal{B}$ uses the Turing machine $M[h]$ where $h = \mathsf{Hash}(\mathsf{hk}, \langle T, w, m \rangle)$. Due to the winning conditions (iii) and (iv), the Turing machine $M[h]$ does not accept h^*w^*, since $M[h]$ accepts h^*w^* only when $h^* = h$, but $h^* \neq h$. Note that similarly to the first case, $\mathcal{B}$ issues a set of signing queries that corresponds to the transition function of $M[h]$ to its own challenger.

In any case, $\mathcal{B}$ only issues a set of signing queries which correspond to some Turing machine M^* (which is either $\overline{M}$ or $M[h]$ mentioned above) that does not accept h^*w^*. From now on we will argue that in these cases there is a signature θ^* in the extracted $\theta^*_{i,j}$'s whose message is not issued by $\mathcal{B}$ as a signing query to its own challenger.

Let $t^*_i = t^*_{i,0} \cdots t^*_{i,\tilde{T}^*+1}$ for all $i \in \{0, \ldots, \tilde{T}\}$. Notice that t^*_0 is the valid initial configuration of M^* with input h^*w^*, and $t^*_{\tilde{T}^*}$ is the configuration whose state is q_{acc}. Then since M^* does not accept h^*w^*, there exists i that satisfies $t^*_i \underset{M^*}{\not\longrightarrow} t^*_{i+1}$. Let i^* be the smallest index satisfying $t^*_{i^*} \underset{M^*}{\not\longrightarrow} t^*_{i^*+1}$. Observe that $t^*_0 \in \Gamma^* \times Q \times \Gamma^*$ and that $t^*_0 \underset{M^*}{\longrightarrow} t^*_1 \underset{M^*}{\longrightarrow} \cdots \underset{M^*}{\longrightarrow} t^*_{i^*}$. It is trivial to check that, if $s \underset{M^*}{\longrightarrow} t$, and $s \in \Gamma^* \times Q \times \Gamma^*$, then $t \in \Gamma^* \times Q \times \Gamma^*$, since a state cannot split into two states as long as $s \underset{M^*}{\longrightarrow} t$. Therefore, we have that $t^*_{i^*} \in \Gamma^* \times Q \times \Gamma^*$ and that $t^*_{i^*} \underset{M^*}{\not\longrightarrow} t^*_{i^*+1}$, and hence we can apply Lemma 1. Lemma 1 ensures that the pair $(t^*_{i^*}, t^*_{i^*+1})$ is not authorized at some position j^* with respect to M^*. Since $\mathcal{B}$ only issues signing queries of the forms that appear in Definition 5, the tuple $(\tau^*, t^*_{i^*,j^*-1}, t^*_{i^*,j^*}, t^*_{i^*,j^*+1}, t^*_{i^*,j^*+2}, t^*_{i^*+1,j^*})$ is never issued as a signing query by $\mathcal{B}$. Thus $\theta^*_{i^*,j^*}$ is a valid forgery for the structure-preserving signature scheme.

To sum up, in any case, that $\mathcal{A}$ satisfies the winning conditions, $\mathcal{B}$ successfully finds a forgery. Therefore, the probability $\Pr[W_4]$ is negligible due to the unforgeability of the structure-preserving signature scheme. $\square$

Finally, we have that all the terms in Eq. (8) are negligible, which implies that $\Pr[W_0]$ is negligible. $\square$

6 Attribute-Based Signature Scheme for Nondeterministic Finite Automata

In this section, we present an attribute-based signature scheme for nondeterministic finite automata. As mentioned in the introduction, this is the first scheme supporting nondeterministic computation as the policy. The syntax and security definitions are similar to those of Turing machines, thus we defer those definitions to the full version.

6.1 Nondeterministic Finite Automata

We give a syntactic definition of finite automata. Let Σ be a finite set of alphabet.

Definition 6. *A nondeterministic finite automaton (NFA) over Σ is defined by the tuple $M = (Q, \delta, q_0, F)$ where: (1) Q is a finite set of states, (2) $\delta \colon Q \times \Sigma \to \mathcal{P}(Q)$ is a transition function, (3) $q_0 \in Q$ is the initial state, and (4) $F \subseteq Q$ is a set of accepting states. We say that a nondeterministic finite automaton $M = (Q, \delta, q_0, F)$ accepts a string $w = w_1 \cdots w_n$ if there exists a sequence $(r_0, r_1, \ldots, r_n)$ of states satisfying (1) $r_0 = q_0$, (2) $\delta(r_{i-1}, w_i) \ni r_i$ for all $i \in \{1, \ldots, n\}$, and (3) $r_n \in F$.*

We remark that the above definition does not allow an automaton to have ε-transitions, i.e., a transition which moves to a new state without reading a symbol. However, it is well known that we can convert any nondeterministic finite automata having ε-transitions into a nondeterministic finite automaton which falls into the above definition, without increasing the number of the states.

For self-containment and ease of understanding, we provide an example of NFA in Fig. 2. This NFA accepts all strings over $\{0, 1\}$ containing a 1 in the third position from the last. It is well known that any NFA can be converted into an equivalent DFA but with an exponential blowup in the number of states (see, for example, [Sip96]). We provide in Fig. 3 such an equivalent DFA to the NFA of Fig. 2. These examples are copied almost verbatim from Sipser's book [Sip96].

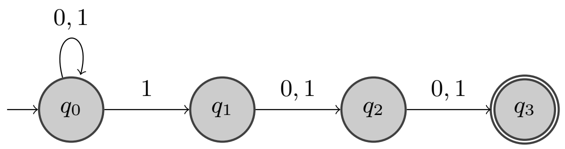

Fig. 2. An example of NFA: it accepts all strings over $\{0, 1\}$ containing a 1 in the third position from the last.

6.2 Notations

We define some notations. For an alphabet Σ, let $\hat{\Sigma}$ be $\Sigma \cup \{-1,-2\}$. We assume that the hash function Hash has as its range $\{-1,-2\}^\ell$ instead of $\{0,1\}^\ell$ to separate the alphabet for hash values from the alphabet for attribute strings. For an NFA $M = (Q, \delta, q_0, F)$, we define an NFA $\hat{M} = (Q, \hat{\delta}, q_0, F)$ over $\hat{\Sigma}$ as follows:

$$\hat{\delta}(q,w) = \begin{cases} \delta(q,w) & (w \in \Sigma), \\ \{q\} & (w \in \{-1,-2\}). \end{cases}$$

6.3 The Scheme

The construction of our scheme is as follows.

$\mathsf{AttrSetup}(1^k, 1^N)$**.** Given a security parameter 1^k and the size 1^N of an alphabet, generate a bilinear group parameter $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g}) \leftarrow \mathcal{G}(1^k)$, a common reference string $\mathsf{crs} \leftarrow \mathsf{WISetup}(\mathsf{gk})$, two key pairs $(\mathsf{vk}_\delta, \mathsf{sk}_\delta) \leftarrow \mathsf{Kg}(\mathsf{gk}, 1^4)$ and $(\mathsf{vk}_F, \mathsf{sk}_F) \leftarrow \mathsf{Kg}(\mathsf{gk}, 1^2)$ of the structure-preserving signature, and a hashing key $\mathsf{hk} \leftarrow \mathcal{H}(1^k)$. Let $\mathsf{pp} \leftarrow (N, \mathsf{gk}, \mathsf{crs}, \mathsf{vk}_\delta, \mathsf{vk}_F, \mathsf{hk})$ and $\mathsf{msk} \leftarrow (\mathsf{sk}_\delta, \mathsf{sk}_F)$ and output $(\mathsf{pp}, \mathsf{msk})$.

$\mathsf{AttrGen}(\mathsf{pp}, \mathsf{msk}, M)$**.** Let $\hat{M}$ be $(Q, \hat{\delta}, q_0, F)$. Choose a random integer $t \leftarrow \mathbb{Z}_p$. For all $q \in Q$, all $w \in \hat{\Sigma}$, and all $q' \in \hat{\delta}(q,w)$ generate a structure-preserving signature $\theta[\![q,w,q']\!]$ on the message

$$([t],[q],[w],[q'])$$

by running $\theta[\![q,w,q']\!] \leftarrow \mathsf{Sign}(\mathsf{vk}_\delta, \mathsf{sk}_\delta, ([t],[q],[w],[q']))$. For all $q \in F$ generate a structure-preserving signature $\rho[\![q]\!]$ on the message

$$([t],[q])$$

by running $\rho[\![q]\!] \leftarrow \mathsf{Sign}(\mathsf{vk}_\rho, \mathsf{sk}_\rho, ([t],[q]))$. Let sk be

$$(M, t, (\theta[\![q,w,q']\!])_{q \in Q, w \in \hat{\Sigma}, q' \in \hat{\delta}(q,w)}, (\rho[\![q]\!])_{q \in F})$$

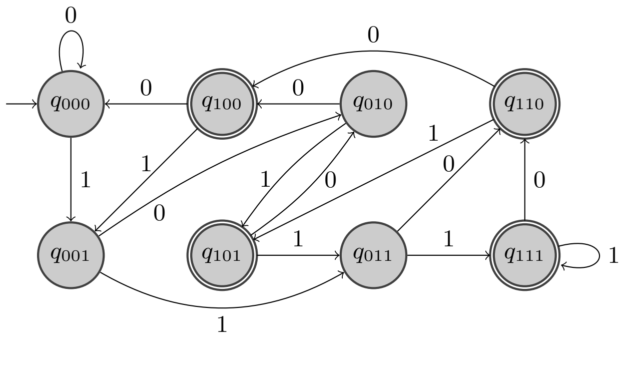

Fig. 3. The smallest DFA that is equivalent to the NFA of Fig. 2.

and output sk.

$\mathsf{AttrSign}(\mathsf{pp}, \mathsf{sk}, w, m)$**.** Let $w_1 \cdots w_n$ be w.

1. Compute $h \leftarrow \mathsf{Hash}(\mathsf{hk}, \langle w, m\rangle)$. Let $\hat{w} = \hat{w}_1 \cdots \hat{w}_{n+\ell}$ be wh.
2. Let $(\hat{q}_0, \hat{q}_1, \ldots, \hat{q}_{n+\ell})$ be one of the sequence of the states that $\hat{M}$ takes when $\hat{M}$ accepts $\hat{w}$.
3. Let $\hat{\theta}_i$ be $\theta[\![\hat{q}_{i-1}, \hat{w}_i, \hat{q}_i]\!]$ for each $i \in \{1, \ldots, n+\ell\}$.
4. Let $\hat{\rho}$ be $\hat{\rho}[\![\hat{q}_{n+\ell}]\!]$.
5. Compute
   - a Groth-Sahai commitment $\psi^{(t)}$ to $[t]$,
   - a Groth-Sahai commitment $\psi_i^{(q)}$ to $[\hat{q}_i]$ for each $i \in \{1, \ldots, n+\ell\}$,
   - a Groth-Sahai commitment $\psi_i^{(\theta)}$ to $\hat{\theta}_i$ for each $i \in \{1, \ldots, n+\ell\}$, and
   - a Groth-Sahai commitment $\psi^{(\rho)}$ to $\hat{\rho}$.
6. Compute a proof $\pi_i^{(\theta)}$ of the equation
$$\mathsf{Verify}(\mathsf{vk}_\delta, ([t], [\hat{q}_{i-1}], [\hat{w}_i], [\hat{q}_i]), \hat{\theta}_i) = 1$$
   for each $i \in \{1, \ldots, n+\ell\}$.
7. Compute a proof $\pi^{(\rho)}$ of the equation
$$\mathsf{Verify}(\mathsf{vk}_F, ([t], [\hat{q}_{n+\ell}]), \hat{\rho}) = 1.$$
8. Let σ be
$$(\psi^{(t)}, \psi_1^{(q)}, \ldots, \psi_{n+\ell}^{(q)}, \psi_1^{(\theta)}, \ldots, \psi_{n+\ell}^{(\theta)}, \psi^{(\rho)}, \pi_1^{(\theta)}, \ldots, \pi_{n+\ell}^{(\theta)}, \pi^{(\rho)})$$
   and output σ.

$\mathsf{AttrVerify}(\mathsf{pp}, w, m, \sigma)$**.** Compute $h \leftarrow \mathsf{Hash}(\mathsf{hk}, \langle w, m\rangle)$ and let $\hat{w}$ be wh. Under this $\hat{w}$ verify all the proofs included in σ where the initial state q_0 is treated as a public constant in the non-interactive proofs. If all the proofs are verified as valid, output 1. Otherwise output 0.

The security of the scheme is postponed to the full version.

7 Conclusion

In this paper, we formalize a new cryptographic primitive, attribute-based signatures for Turing machines. We also present an efficient instantiation of this primitive using the Groth-Sahai proof system, a structure-preserving signature scheme, and a collision-resistant hash function family. In addition, we present an attribute-based signature scheme for NFA, which is more efficient than our first scheme, while less expressive. These two schemes provide a trade-off between efficiency and expressiveness.

Acknowledgment. The first author is supported by JSPS KAKENHI Grant Number 18K18055. The second author was partially supported by JST CREST Grant Number JPMJCR1302 and JSPS KAKENHI Grant Number 17J05603.

References

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for Turing machines. In *TCC (A1)*, volume 9562 of *LNCS*, pages 125–153. Springer, 2016.

[BF14] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In *PKC 2014*, volume 8383 of *LNCS*, pages 520–537. Springer Berlin Heidelberg, 2014.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer Berlin Heidelberg, 2014.

[DDM17] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Attribute-based signatures for Turing machines. Cryptology ePrint Archive, Report 2017/801, 2017. `http://eprint.iacr.org/2017/801`.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In *CRYPTO (2)*, volume 8043 of *LNCS*, pages 536–553. Springer, 2013.

[GS12] Jens Groth and Amit Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, October 2012.

[KPW15] Eike Kiltz, Jiaxin Pan, and Hoeteck Wee. Structure-preserving signatures from standard assumptions, revisited. In *CRYPTO 2015, Part II*, volume 9216 of *LNCS*. Springer Berlin Heidelberg, 2015.

[MPR11] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In *CT-RSA 2011*, volume 6558 of *LNCS*, pages 376–392. Springer Berlin Heidelberg, 2011.

[NP15] Mridul Nandi and Tapas Pandit. On the power of pair encodings: Frameworks for predicate cryptographic primitives. Cryptology ePrint Archive, Report 2015/955, 2015. `http://eprint.iacr.org/`.

[OT11] Tatsuaki Okamoto and Katsuyuki Takashima. Efficient attribute-based signatures for non-monotone predicates in the standard model. In *PKC 2011*, volume 6571 of *LNCS*, pages 35–52. Springer Berlin Heidelberg, 2011.

[SAH16] Yusuke Sakai, Nuttapong Attrapadung, and Goichiro Hanaoka. Attribute-based signatures for circuits from bilinear map. In *PKC 2016*, volume 9614 of *LNCS*, pages 283–300. Springer Berlin Heidelberg, 2016.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[SSN09] Siamak F. Shahandashti and Reihaneh Safavi-Naini. Threshold attribute-based signatures and their application to anonymous credential systems. In *AFRICACRYPT 2009*, volume 5580 of *LNCS*, pages 198–216. Springer Berlin Heidelberg, 2009.

[TLL14] Fei Tang, Hongda Li, and Bei Liang. Attribute-based signatures for circuits from multilinear maps. In *ISC 2014*, volume 8783 of *LNCS*, pages 54–71. Springer Berlin Heidelberg, 2014.