Expiration and Revocation of Keys for Attribute-based Signatures [⋆]

Stephen R. Tate[1] and Roopa Vishwanathan[2]

[1] Department of Computer Science, UNC Greensboro, Greensboro, NC 27402
`srtate@uncg.edu`
[2] Department of Computer Science, SUNY Poly, Utica, NY 13502
`vishwar@sunyit.edu`

Abstract. Attribute-based signatures, introduced by Maji *et al.*, are signatures that prove that an authority has issued the signer "attributes" that satisfy some specified predicate. In existing attribute-based signature schemes, keys are valid indefinitely once issued. In this paper, we initiate the study of incorporating time into attribute-based signatures, where a time instance is embedded in every signature, and attributes are restricted to producing signatures with times that fall in designated validity intervals. We provide three implementations that vary in granularity of assigning validity intervals to attributes, including a scheme in which each attribute has its own independent validity interval, a scheme in which all attributes share a common validity interval, and a scheme in which sets of attributes share validity intervals. All of our schemes provide anonymity to a signer, hide the attributes used to create the signature, and provide collusion-resistance between users.

Keywords: Attribute-Based Signatures, Key Revocation, Key Expiration

1 Introduction

In some situations, users authenticate themselves based on credentials they own, rather than their identity. Knowing the identity of the signer is often less important than knowing that a user possesses certain credentials or attributes, e.g., "over 21 years old," or "computer science major." This form of authentication is ideal for loosely-knit situations where anonymity and unforgeability are desired, and one needs to be sure that users cannot collude to combine attributes from each other to satisfy authentication challenges. To this end, Maji *et al.* [10] introduced *attribute-based signatures* as a primitive that allows users to sign messages anonymously using a combination of their attributes. The parties involved in

[⋆] A shorter version of this paper appeared at the *29th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2015)*. This material is based upon work supported by the National Science Foundation under Grant No. 0915735.

attribute-based signatures are a signature trustee (ST), an attribute-issuing authority (AIA), and potentially many signers and verifiers. The signature trustee acts as a globally trusted source that sets the global system parameters correctly (e.g., honestly generates a common reference string), and the attribute-issuing authority, which is trusted in more limited ways, issues signing keys for attributes to users. Although the AIA knows the signing keys and attributes of all users, it cannot tell which attributes have been used in a given valid signature, and hence cannot identify the signatures made by any user and/or link signatures made by a single user.

In the original work of Maji *et al.* [10], the basic scheme uses attributes that do not have any time restrictions on validity – once an attribute is issued, it is good forever (or at least as long as the global public verification key is valid). Maji *et al.* [10] informally describe some ideas for attribute expiration and revocation, but these issues are simply mentioned in passing. In this paper, we initiate a careful study of restricting attribute validity in attribute-based signature schemes, providing a formal framework as well as implementations that are significantly more efficient than those that were suggested in earlier work.

A user who receives a key for a set of attributes from an AIA can sign a message with a predicate that is satisfied by their attributes. Predicates, or claim predicates, are Boolean expressions over a set of attributes, and satisfying a predicate involves supplying a valid combination of attributes such that the Boolean expression evaluates to true. Signature verification tests if the signature was performed by a user with a satisfying set of attributes, without needing to know the signer's attributes or identity. The main interesting properties of attribute-based signatures are *anonymity* of both the signer's identity and specific attributes used in generating the signatures, even if one has full information about which users were issued which attributes, and *collusion-resistance*, where two or more users cannot pool their attributes together to satisfy a predicate that they cannot individually satisfy. Note that since traditional digital signatures are verified using a user-specific public key, such a signature cannot provide the anonymity property required of an attribute-based signature.

In real-world situations, a user may be issued a time-limited attribute that has a well-defined validity period consisting of an issue date and expiry date. Since explicit revocation is not possible in the anonymous setting of attribute-based signatures, attributes can be used until they expire, forcing frequent expiration. As a simple example that motivates revocation, an organization could issue `Employee` attributes to its employees, which they use for authentication. Once an employee leaves the organization, they should no longer be able to authenticate using their `Employee` attribute. In addition to the expiry date, it is also important to check the issue date of an attribute, or the start of validity. Consider an organization where employees can anonymously certify or sign internal company documents, as long as they have valid credentials. Alice is an employee that joined the organization in March 2012, and was issued an `Employee` attribute. She should not be able to use this attribute to produce valid signatures over documents for February 2012, or any time before her start date. This property is referred to as *forward security*, in the signature literature.

In this paper we take an abstract view of time, with concrete instantiations for traditional notions of time (which we call "clock-based time") and a trusted server instantiation (which we call "counter-based time") which allows for instant revocation by incrementing a counter on a trusted time server.

Related Work. Attribute-based signature revocation was briefly mentioned by Maji *et al.* [10], but they don't give any specifics on how attribute sets can incorporate signing key revocation or attribute set revocation. Escala *et al.* [4] introduce schemes for revocable attribute-based signatures, but in their paper, "revocability" refers to revoking the anonymity of a user who created a signature (revealing their identity), and not revoking signing keys or attribute sets. Their revoke function is run by a party whose role is similar to that of a group manager in group signatures, and takes in an attribute-based signature, some public parameters and state information, and outputs the identity of the user who created the signature. Li *et al.* [9], Shahandashti and Safavi-Naini [15], and Herranz *et al.* [7] present attribute-based signature schemes, but do not deal with attribute and key revocation and expiry. Okamoto and Takashima [11, 12] propose efficient attribute-based signature schemes which support a rich range of predicates, and do not require any trusted setup, respectively, but do not consider revocation.

In this paper we focus exclusively on authentication and attribute-based *signatures*. A significant amount of work has been done recently in the area of attribute-based *encryption* [8, 5, 13, 17, 3], but those techniques do not carry over into the signature realm and can be viewed as orthogonal to our work.

Our Contributions. The contributions of this paper are briefly summarized as follows:

- Extension of attribute-based signature definitions to support attribute expiration;
- A generic notion of time that includes instantiations for not only traditional ("clock-based") time, but also a trusted counter based notion that allows instant revocation;
- Key-update mechanisms that allow efficient extension of issued attribute sets; and
- Three implementations that vary in granularity of associating intervals with attributes and have various efficiency trade-offs.

2 Definitions

In this section we develop definitions for a time-aware attribute-based signature (ABS) scheme, and since the motivation is to support attribute expiration for revocation, we call this a Revocable Attribute-Based Signature scheme, or "RABS." The starting point for our definition is the ABS definition from Maji *et al.* [10]. At the core of any attribute-based scheme are the attributes, defined by a universe of attributes $\mathbb{A}$. An attribute $a \in \mathbb{A}$ is a generic name (e.g., `Employee`), and when we say that an attribute is "issued" to a user we are really talking about a private signing key associated with that attribute being generated by the AIA and provided to the user. Keys are associated with sets of attributes, and each instance of a attribute set signing key has a public identifier *pid* (users do not have individual public keys).

Attribute-based signatures are made with respect to a predicate Υ over attributes. For RABS we use monotone span programs to specify Υ, the same as Maji *et al.* [10]. A span program $\Upsilon = (\mathbf{M}, a)$ consists of an $\ell \times k$ matrix $\mathbf{M}$ over a field $\mathbf{F}$, with a labeling function $a : [\ell] \to \mathbb{A}$ that associates each of the ℓ rows of $\mathbf{M}$ with an attribute. The monotone span program is satisfied by a set of attributes $\mathcal{A} \subseteq \mathbb{A}$, written $\Upsilon(\mathcal{A}) = 1$, if and only if

$$\exists \boldsymbol{v} \in \mathbf{F}^{1\times\ell} \ : \ \boldsymbol{v}\mathbf{M} = [1, 0, 0, \cdots, 0] \text{ and } (\forall i : v_i \neq 0 \Longrightarrow a(i) \in \mathcal{A}) \qquad (1)$$

Another way to view this is that the monotone span program is satisfied if and only if $[1, 0, 0, \cdots, 0]$ is in the span of the row vectors corresponding to the attributes held by the user.

2.1 Time and Validity Intervals

In this paper, times can be drawn from any partially ordered set $(T, \leq)$. There is a trusted time source that can report an authenticated "current time" to any party in the system, and it is required that the sequence of reported times be a totally ordered subset of T. Time intervals are specified as closed intervals such as $[t_s, t_e]$, where $t_s \leq t_e$, and a time value t is said to be in the interval (written $t \in [t_s, t_e]$) if $t_s \leq t \leq t_e$. In RABS, attributes have associated *validity intervals*, so if $a \in \mathbb{A}$ is a non-time-specific attribute, in RABS we would typically refer to $(a, [t_s, t_e])$ meaning that this attribute is valid at all times $t \in [t_s, t_e]$. As a more compact notation, we will sometimes use ι to denote an interval, so a time-specific attribute might be denoted (a, ι).

RABS signatures include a specific time $t \in T$ in the signature, so we typically write a RABS signature as $\sigma = (t, \phi)$, and we call this a "time-t signature." A valid time-t signature can only be made by a user who has been issued attributes $(a_i, [t_{s_i}, t_{e_i}])$ for $i = 1, \ldots, n$, such that $\Upsilon(\{a_i \mid i = 1, \ldots, n\}) = 1$ and $t \in [t_{s_i}, t_{e_i}]$ for all $i = 1, \ldots, n$. While it is tempting to refer to a "signature made at time t," it is clearly impossible to restrict when a signature is actually created — a time t signature could in fact be created at any time, as long as the signer holds (possibly old) keys that were valid at time t. Note that in one prominent application, a real-time authentication scenario in which a challenger provides the current time t and a nonce to the prover, who is then required to produce a time-t signature over the nonce, it *does* make sense to think of this as a signature being made at time t.

The everyday notion of time (which we will refer to as "clock-based time") easily meets these requirements, where each element of T is actually an interval defined with respect to some level of granularity of time, such as seconds, days, weeks, or months. For example, if T were the set of all months, then there might be a time value such as $t = \texttt{2014-March}$. These times form a totally ordered set, and larger intervals can be specified such as $[\texttt{2014-March}, \texttt{2014-June}]$. In clock-based time, we assume that there are a limited set of *standard validity intervals* that are used for attributes. For example, if T contains individuals days, then we could have standard validity intervals that represent monthly, weekly, or daily intervals, so a single-day time $t = \texttt{2014-Jan-09}$ could be in standard validity intervals $[\texttt{2014-Jan-01}, \texttt{2014-Jan-31}]$ (monthly), $[\texttt{2014-Jan-06}, \texttt{2014-Jan-12}]$ (weekly), or $[\texttt{2014-Jan-09}, \texttt{2014-Jan-09}]$ (daily).

As an alternative to clock-based time, we can let T be a set of vectors over integer counter variables, where two times $t_1 = \langle t_{1,1}, t_{1,2}, \cdots, t_{1,k} \rangle$ and $t_2 = \langle t_{2,1}, t_{2,2}, \cdots, t_{2,k} \rangle$ are compared by

$$t_1 \leq t_2 \quad \iff \quad \forall i \in 1, \cdots, k, \quad t_{1,i} \leq t_{2,i}.$$

We call this notion "counter-based time," and in this case the trusted time source could maintain a set of monotonic counters for each vector entry so that counters could be independently incremented on demand. While the set T is only partially ordered, since the individual counters are monotonic, the trusted time source would never output two times that are incomparable, such as $\langle 1, 2 \rangle$ and $\langle 2, 1 \rangle$.

While using vectors of counters for time requires more space to specify a time, it is a significantly more powerful notion. In particular, a counter can correspond to a set of attributes, and then when an attribute in that set needs to be revoked the counter can be incremented on demand. This allows for immediate expiration of issued attributes, rather than having to wait until the end of the current time period (e.g., the end of the month) as you would have to do with clock-based time. We discuss in more depth some properties and uses of counter-based time in Section 5.

2.2 Basic Techniques

A fundamental part of making or verifying a time-t signature in RABS implementations is the conversion of a span program $\Upsilon = (\mathbf{M}, a)$ that does not take time into consideration into a span program $\Upsilon' = (\mathbf{M}', a')$ that includes requirements that t is in the validity interval of all attributes used to satisfy Υ. The precise form of our transformation depends on the specific implementation, so we will introduce these transformations in later sections. Recall that "issuing an attribute set" means providing a user with a signing key that corresponds to a set of attributes. We write a generic secret key as SK, and we can also add designations to this secret key to indicate that it has certain properties. For example, $SK_{\mathcal{A}}$ refers to a signing key for the specified set of attributes, $SK^t_{\mathcal{A}}$ refers to a signing key in which all attributes in $\mathcal{A}$ are valid at time t.

Another novel idea that we introduce in this paper is the idea of a "key change." Some RABS operations can be accomplished with a small change to an already-issued signing key, so rather than communicate a full and nearly-identical key we communicate a Δ which describes how to change the existing key into a new key. Consider the following situation: a user has been issued a large attribute set $\mathcal{A}$, with hundreds of attributes — this is a very large signing key. At some point, these attributes expire and we wish to renew or reissue them for a new time period. Our goal then is to produce a small, compact Δ that describes changes to an existing signing key, say $SK^t_{\mathcal{A}}$, so that we can apply this Δ to update the key. While the precise format of Δ depends on a specific implementation, our implementations treat Δ as a sequence of commands such as $\langle \textsc{New}, id, SK\rangle$ for replacing a component of a key, identified as id, with a new key SK. All of these notions are combined to yield the definition of a RABS scheme, given in Definition 1.

Definition 1. *A Revocable ABS (RABS) scheme has the following functions:*

- $\mathsf{RABS.TSetup}(1^\lambda) \rightarrow (TPK, TSK)$*: Run by the signature trustee to generate a common public key or reference string,* TPK*, and a secret key* TSK*.*
- $\mathsf{RABS.Register}(TSK, uid) \rightarrow \tau$*: Run by the signature trustee to register a user.* τ *can bind user-specific parameters chosen by the trustee to a user id (uid). For example, in one of Maji* et al.*'s implementations,* τ *consists of some trustee and user-specific public parameters, signed by the trustee.*
- $\mathsf{RABS.ASetup}(TPK, 1^\lambda) \rightarrow (APK, ASK)$*: Run by the attribute-issuing authority (AIA) to generate a keypair* (APK, ASK)*.*
- $\mathsf{RABS.AttrGen}(\tau, TPK, ASK, \mathcal{A} = \{(a_1, \iota_1), \ldots, (a_u, \iota_u)\}) \rightarrow (SK_{\mathcal{A}}, pid, \psi)$*: Run by the AIA to issue a signing key for time-specified attribute set* $\mathcal{A}$ *for a user identified in* τ*. We assume that that AIA has verified* τ *before using this function. Outputs include the private signing key* $SK_{\mathcal{A}}$ *to be given to the user, the public identifier pid for this key, and* ψ *which is the user-specific state that is maintained by the AIA as required to work with this set efficiently on future requests, such as* Reissue *and* Extend*.*
- $\mathsf{RABS.Sign}(TPK, APK, SK_{\mathcal{A}}, m, \Upsilon, t) \rightarrow \sigma$*: Run by a user that possesses signing key* $SK_{\mathcal{A}}$ *for attributes* $\mathcal{A}$ *which are all valid at time* t*, to produce a time-*t *signature on message* m*. Note that the time* t *is embedded in the signature, so we will sometimes write* $\sigma = (t, \phi)$ *to make the time explicit.*
- $\mathsf{RABS.Ver}(TPK, APK, m, \Upsilon, \sigma) \rightarrow v \in \{\text{"accept"}, \text{"reject"}\}$*: Run by a verifier to validate signature* $\sigma = (t, \phi)$*. Verifies that the signature was made by some user that was issued attributes* $\mathcal{A}$ *that were valid at time* t *such that* $\Upsilon(\mathcal{A}) = 1$*.*
- $\mathsf{RABS.ReIssue}(\tau, \psi, pid, TPK, ASK, \mathcal{A} = \{(a_1, \iota_1), \ldots, (a_u, \iota_u)\}) \rightarrow (\Delta, \psi')$*: Run by the AIA to extend the valid time intervals for all of the associated attributes. For this function, pid should reference an existing issued attributed set for an* $\mathcal{A}' = \{(a_1, \iota'_1), \ldots, (a_u, \iota'_u)\}$ *with the same attributes at different (typically earlier) time intervals, and the output includes a signing key update description* Δ *and updated AIA state information* ψ'*.* Δ *compactly describes how to update the current signing key, and is sent to the user.*
- $\mathsf{RABS.Extend}(\tau, \psi, pid, TPK, ASK, \mathcal{A}' = \{(a_1, \iota_1), \ldots, (a_u, \iota_u)\}) \rightarrow (\Delta, \psi')$*: Run by the attribute-issuing authority, to add new attributes to the already-issued set pid, which currently covers attribute set* $\mathcal{A}$*. The outputs are the same as* RABS.ReIssue*, where* Δ *contains information that allows the user to update their signing key so that it covers extended attribute set* $\mathcal{A} \cup \mathcal{A}'$*.*

– $\mathsf{RABS.Update}(TPK, APK, \Delta, SK) \rightarrow (SK')$*: This is a deterministic algorithm that is run by the user, where the user takes in an old signing key, an update description* Δ *generated by the attribute authority, and outputs a new signing key* SK'.

3 Threat Model and Security Properties

In the attribute-based signature (ABS) model, the adversary could either be a user who was issued attributes and keys valid for a fixed time period, or could be an external party that compromised the user's attributes and keys. Additionally, the attribute issuing authority could itself be considered an adversary colluding with a malicious user and/or external parties. We note that in our model, as well as in previous work in attribute-based signatures, the attribute-issuing authorities are considered malicious only in the sense that they will try to violate the anonymity of a user signing a message, and they are still trusted to correctly distribute attributes and signing keys among users. In particular, in the ABS model, one generally does not consider issues such as the attribute authorities unilaterally issuing (or re-issuing) signing keys, and using them to sign on behalf of a user. Furthermore, the signature trustee is considered to be a trusted party.

We now give a formal definition of security for any RABS scheme. Consider two adversaries, $\mathfrak{S}_1$ and $\mathfrak{S}_2$. Both adversaries know general system parameters, such as the set T, and are given public keys of the signature trustee and attribute authority when they are generated. The goal of $\mathfrak{S}_1$ is to guess which attributes were used to satisfy Υ, and the goal of $\mathfrak{S}_2$ is to forge a signature that passes verification, despite not having been issued satisfying attributes that are good at the time t embedded in the signature. The definition follows, and is derived from the ABS definitions of Maji *et al.* [10] — more discussion is in that paper.

Definition 2. *A secure RABS scheme possesses the following properties:*

1. ***Correctness:*** *A RABS scheme is said to be correct if for all* $(TPK, TSK) \leftarrow \mathsf{RABS.TSetup}(1^\lambda)$, *all* $(APK, ASK) \leftarrow \mathsf{RABS.ASetup}(TPK, 1^\lambda)$, *all messages* m, *all attribute sets* $\mathcal{A}$, *all claim-predicates* Υ *such that* $\Upsilon(\mathcal{A}) = 1$, *all keys* $(SK_\mathcal{A}, pid, \psi) \leftarrow \mathsf{RABS.AttrGen}(\tau, TPK, ASK, \mathcal{A})$, *and all signatures* $\sigma \leftarrow \mathsf{RABS.Sign}(TPK, APK, SK_\mathcal{A}, m, \Upsilon, t)$, *we have* $\mathsf{RABS.Ver}(TPK, APK, m, \Upsilon, \sigma) =$ *"accept".*
2. ***Perfect Privacy:*** *A RABS scheme has perfect privacy if, for* TPK *that are all honestly generated with* $\mathsf{RABS.TSetup}$, *all* APK, *all attribute sets* $\mathcal{A}_1$ *and* $\mathcal{A}_2$, *all* $SK_1 \leftarrow \mathsf{RABS.AttrGen}(.., \mathcal{A}_1)$ *and* $SK_2 \leftarrow \mathsf{RABS.AttrGen}(.., \mathcal{A}_2)$, *and all* Υ *such that* $\Upsilon(\mathcal{A}_1) = \Upsilon(\mathcal{A}_2) = 1$, *the distributions* $\mathsf{RABS.Sign}(TPK, APK, SK_1, m, \Upsilon, t)$ *and* $\mathsf{RABS.Sign}(TPK, APK, SK_2, m, \Upsilon, t)$ *are identical.*
3. ***Existential Unforgeability:*** *A RABS scheme is existentially unforgeable if adversary* $\mathfrak{S}_2$, *given black-box access to a RABS oracle* $\mathcal{O}$, *has negligible probability of winning the following game:*
   – *Run* $(TPK, TSK) \leftarrow \mathsf{RABS.TSetup}(1^\lambda)$, *and* $(APK, ASK) \leftarrow \mathsf{RABS.ASetup}(TPK, 1^\lambda)$. TPK, APK *are given to* $\mathfrak{S}_2$.

- $\mathfrak{S}_2$ *runs a probabilistic polynomial time algorithm in which it can make queries to registration oracle* $\mathcal{O}^{\mathsf{RABS.Register}(\mathsf{TSK},\cdot)}$*, key generation and modification oracles* $\mathcal{O}^{\mathsf{RABS.AttrGen}(\cdot,\cdot,\mathsf{ASK},\cdot)}$*,* $\mathcal{O}^{\mathsf{RABS.ReIssue}(\cdot,\cdot,\cdot,\cdot,\mathsf{ASK},\cdot)}$*, and* $\mathcal{O}^{\mathsf{RABS.Extend}(\cdot,\cdot,\cdot,\cdot,\mathsf{ASK},\cdot)}$.
- $\mathfrak{S}_2$ *outputs* (m', Υ', σ').

$\mathfrak{S}_2$ *succeeds if* $\mathsf{RABS.Ver}(TPK, APK, m', \Upsilon', \sigma') =$ *"accept",* $\mathcal{O}^{\mathsf{RABS.Sign}}$ *was never queried with* (m', Υ')*, and* $\Upsilon'(\mathcal{A}) = 0$ *for all* $\mathcal{A}$ *queried to* $\mathcal{O}^{\mathsf{RABS.AttrGen}}$.

4 Implementations

In this section, we present several implementations for RABS. The implementations differ in how attributes use validity intervals: each attribute can be assigned a validity interval that is independent of the others; all attributes can share the same validity interval; or attributes can be grouped into sets that share the same validity interval. All of our implementations are built on top of a secure non-time-specific ABS scheme — for Implementations 1 and 2, any ABS scheme that satisfies the security properties in Maji *et al.* [10] will work, but Implemention 3 has more strict requirements, as described in Section 4.4.

4.1 Implementation 1: Independent Validity Intervals

For this implementation, each attribute is assigned a validity interval that is independent of any other validity interval used by any other attribute. To accomplish this, we incorporate the validity interval into the attribute name. For example, a time-specific attribute (`Employee`, [`2014-Jan-06`, `2014-Jan-12`]) would be named `Employee-2014-Jan-06-2014-Jan-12`.

Using the notion of standard validity intervals from Section 2.1, consider a time t that is contained in k standard validity intervals: $\iota_1, \ldots, \iota_k$. Viewing the condition Υ as a Boolean formula, when calling the $\mathsf{RABS.Sign}$ function to make a time-t signature we would first change every occurrence of an attribute a in the Boolean formula to a disjunction of all attributes that incorporate a standard time interval that includes time t. In other words, an occurrence of attribute a in the Boolean formula would be replaced with $(a\text{-}\iota_1 \vee \cdots \vee a\text{-}\iota_k)$. Viewing the condition $\Upsilon = (\mathbf{M}, a)$ as a monotone span program, since each row $i = 1, \ldots, \ell$ of the original $\mathbf{M}$ corresponds to attribute $a(i)$, we simply duplicate this row k times, and map the time-specific attributes to these rows. In the example of standard validity intervals from Section 2.1, if $a(i) =$ `Employee` then we duplicate that row 3 times and map the monthly, weekly, and daily validity intervals to these 3 rows. We will refer to the expanded matrix as $\Upsilon' = (\mathbf{M}', a)$.

This implementation is an obvious way of adding support for validity intervals to attribute-based signatures, and the overhead for signatures is not large: If $\mathbf{M}$ is $\ell \times k$ and there are c time intervals that are valid for each attribute (e.g., $c = 3$ in our example above), then the resulting Υ' used in making signatures includes a $c\ell \times k$ matrix $\mathbf{M}'$. However, this implementation has a major disadvantage, which was the motivation for this work: the AIA has a motivation to expire attributes frequently so that attributes can be effectively revoked without a long delay, but most of the time a user's set of attributes will simply be reissued for the following time interval. This implementation requires the AIA to frequently reissue all attributes for every user in this situation, and if users hold lots of fine-grained attributes this is very expensive.

Theorem 1. *Given a secure non-time based ABS scheme, Implementation 1 is a secure RABS scheme.*

Proof. First, note that, given a time t, Υ' is satsfied by set of time-specific attributes $\mathcal{A}' = \{(a_1, \iota_1), \ldots, (a_s, \iota_s)\}$ if and only if $t \in \iota_i$ for all $i = 1, \ldots, s$ and $\Upsilon(\{a_1, \ldots, a_s\}) = 1$. The Correctness and Perfect Privacy properties of Implementation 1 follow directly from this fact and the corresponding properties of the underlying ABS scheme. For Existential Unforgeability, note that an adversary playing against a RABS oracle can be easily converted into an adversary playing against an ABS oracle since none of the RABS-to-ABS conversion uses oracle-only secrets such as TSK, ASK, or signing keys for Sign oracle calls. As a result, any RABS adversary that wins with non-negligible probability can become a ABS adversary that wins with non-negligible probability — but since the ABS scheme is existentially unforgeable, this is impossible. □

4.2 Validity Attributes

The next two implementations rely on a special type of attribute called a *validity attribute*. When a validity attribute is issued as part of an attribute set it indicates that all attributes in the set are valid during the validity attribute's interval. A single set may contain validity attributes with different validity intervals, making the set valid for multiple time intervals (which is essential for our efficient $\mathsf{RABS.ReIssue}$ operation), but attribute sets that are separately issued cannot be combined due to the non-collusion property of the underlying ABS scheme. In other words, a user could not take a current validity attribute and combine it with an expired, previously-issued set of attributes, even if all of these attributes had been properly issued to the same user.

There can be multiple, distinct validity attribute names, and we denote the full set of validity attribute names as $\mathbb{V}$. Different regular attributes may use different validity attributes from $\mathbb{V}$, but each regular attribute has a single validity attribute that it can use. We define a map $v : \mathbb{A} \to \mathbb{V}$ that gives the validity attribute $v(a)$ that can be used for attribute $a \in \mathbb{A}$. We define the following set to restrict attention to validity attributes that can be used for a specific attribute at a specific time $t \in T$:

$$\mathbb{V}_{a,t} = \{(v(a), \iota) \,|\, \iota = [t_s, t_e] \text{ is a standard validity interval with } t \in [t_s, t_e]\}$$

If the set of standard validity intervals for any time t is small, as we expect it would be in practical applications, then $|\mathbb{V}_{a,t}|$ will be bounded by a small constant.

Incorporating Validity Attributes into the Monotone Span Program. When creating or verifying a time-t signature using validity attributes, we modify a non-time-specific monotone access program $\varUpsilon = (\mathbf{M}, a)$, where $\mathbf{M}$ is an $\ell \times k$ matrix, to create a time-specific monotone access program $\varUpsilon' = (\mathbf{M}', a')$ for time t as follows. For each row $i = 1, \cdots, \ell$ we add a single new column, which we refer to as column $nc(i)$, and add a new row for each $v \in \mathbb{V}_{a(i),t}$ which we refer to as row $nr(i, v)$. Each new row $nr(i, v)$ contains a 1 in column $nc(i)$ and zeroes in all other columns. In addition to those 1's, new column $nc(i)$ also has a 1 in row i, and zeroes in all other entries. To expand the labeling function a to a', we map each new row $nr(i, v)$ to validity attribute v. We call this transformation of span programs T, and since it depends on both the original monotone span program $\varUpsilon$ and the time t, we can denote this as $\varUpsilon' = T(\varUpsilon, t)$.

The following Lemma shows that a user can satisfy $\varUpsilon'$ if and only if she has been issued attributes that satisfy the non-time-specific $\varUpsilon$ as well as validity attributes that validate each attribute she uses at time t.

Lemma 1. *Given a non-time-specific monotone access program $\varUpsilon$, the constructed time-specific monotone access program $\varUpsilon' = T(\varUpsilon, t)$, and two sets of attributes $\mathcal{A} \subseteq \mathbb{A}$ and $\mathcal{V} \subseteq \mathbb{V}$, $\varUpsilon'(\mathcal{A} \cup \mathcal{V}) = 1$ if and only if $\varUpsilon(\mathcal{A}) = 1$ and for every $a \in \mathcal{A}$ there exists a $v \in \mathcal{V}$ such that $v \in \mathbb{V}_{a,t}$.*

Proof. Let $\varUpsilon = (\mathbf{M}, a)$ and $\varUpsilon' = (\mathbf{M}', a')$, where $\mathbf{M}$ is $\ell \times s$ and $\mathbf{M}'$ is $\ell' \times k$. For the first direction of the proof, let $\mathcal{A}$ and $\mathcal{V}$ be as described in the final clause of the lemma, so that $\varUpsilon(\mathcal{A}) = 1$, and for every $a \in \mathcal{A}$ there is a $v \in \mathcal{V}$ such that $v \in \mathbb{V}_{a,t}$. We will show that $\varUpsilon'(\mathcal{A} \cup \mathcal{V}) = 1$. Since $\varUpsilon(\mathcal{A}) = 1$, there must be some vector $\boldsymbol{w} \in \mathbf{F}^{1\times\ell}$ such that $\boldsymbol{w}\mathbf{M} = [1, 0, \cdots, 0]$, where every non-zero coordinate w_i corresponds to an attribute $a(i) \in \mathcal{A}$. Constructing a $\boldsymbol{w}' \in \mathbf{F}^{1\times\ell'}$ so that $\boldsymbol{w}'\mathbf{M}' = [1, 0, \cdots, 0]$ is then fairly straightforward: The first ℓ coordinates of $\boldsymbol{w}$ are copied to $\boldsymbol{w}'$, and for each original row $i \in \{1, \ldots, \ell\}$ we pick one $v \in \mathbb{V}_{a(i),t}$ and set coordinate $w'_{nr(i,v)} = -w_i$. All other $\boldsymbol{w}'$ coordinates are zero. It is easy to verify that the first k columns in $\boldsymbol{w}'\mathbf{M}'$ keep the same value as in $\boldsymbol{w}\mathbf{M}$ and each new column has two coordinates that exactly cancel each other out, so the result is that $\boldsymbol{w}'\mathbf{M}' = [1, 0, \cdots, 0]$. Therefore, $\varUpsilon(\mathcal{A}') = 1$.

For the other direction of the "if and only if," let $\mathcal{A}'$ be a set of attributes such that $\varUpsilon'(\mathcal{A}') = 1$, and partition $\mathcal{A}'$ into sets $\mathcal{A}$ and $\mathcal{V}$ for the original attributes and validity attributes, respectively. Then there must be a $\boldsymbol{w}' \in \mathbf{F}^{1\times\ell'}$ such that $\boldsymbol{w}'\mathbf{M}' = [1, 0, \cdots, 0]$ and each $w_i \neq 0$ corresponds to an attribute $a(i) \in \mathcal{A}'$. Taking the first ℓ coordinates of $\boldsymbol{w}'$ to form $\boldsymbol{w}$, and noting that the first ℓ columns of $\mathbf{M}'$ have zeroes in rows $\ell + 1$ and higher, it follows that $\boldsymbol{w}\mathbf{M} = [1, 0, \cdots, 0]$ and so $\varUpsilon(\mathcal{A}) = 1$. Next, consider column $nc(i)$ that was added when $\mathbf{M}'$ was created, which we will denote as $\mathbf{M}'_{\cdot,nc(i)}$. Since $\boldsymbol{w}'\mathbf{M}' = [1, 0, \cdots, 0]$, we know that $\boldsymbol{w}' \cdot \mathbf{M}'_{\cdot,nc(i)} = 0$. Furthermore, since $\mathbf{M}'_{\cdot,nc(i)}$ is zero everywhere except row i and rows $nr(i, v)$, which are 1's, if $w'_i \neq 0$, meaning $a(i) \in \mathcal{A}$, and the dot product is non-zero, then at least one of the $\boldsymbol{w}'$ coordinates corresponding to rows $nr(i, v)$ must also be nonzero. Let v be such that $w'_{nr(i,v)} \neq 0$, and so $v \in \mathcal{V}$ and $v \in \mathbb{V}_{a(i),t}$, which completes the proof. $\square$

Transformation T results in an expanded matrix $\mathbf{M}'$ that has $\ell + \sum_{i=1}^{\ell} |\mathbb{V}_{a(i),t}|$ rows and $s + \ell$ columns. We expect that in practice the set of possible validity intervals at time t will be a fixed set that does not depend on the attribute, so we can write this simply as $(|\mathbb{V}_t|+1)\ell$ rows by $s+\ell$ columns.

4.3 Implementation 2: Common Validity Interval

In this implementation, there is only a single validity interval that applies to all issued attributes, and so all attributes will share that validity interval. The big advantage that we gain is that an entire set of attributes can be reissued for a new validity interval by just issuing a single new validity attribute to the user, making the "common case" much more efficient than Implementation 1. Furthermore, implementation is still straightforward using any standard non-time based attribute-based signature scheme, and the basic setup and key management functions (TSetup, $\mathsf{Register}$, ASetup, Update) carry over without modification. The remaining operations are defined below. Most operations are simple extensions based on the transformation from Lemma 1, but the Extend operation addresses one subtle issue which is explained further in the following text.

– $\mathsf{RABS.AttrGen}(\tau, TPK, ASK, \mathcal{A} = \{(a_1,\iota_1),\ldots,(a_u,\iota_u)\})$: This operation fails if all ι_i's are not the same standard validity interval $[t_s,t_e]$. Otherwise, define validity attribute $v = (\mathsf{validity}, [t_s,t_e])$ and call $\mathsf{ABS.AttrGen}(\tau, TPK, ASK, \mathcal{A}' = \{v, a_1, a_2, \ldots, a_u\})$ from the underlying standard attribute-based signature scheme, returning the result (SK, pid, ψ).
– $\mathsf{RABS.Sign}(TPK, APK, SK_{\mathcal{A}}, m, \Upsilon, t)$: Transformation T is used to compute $\Upsilon' = T(\Upsilon, t)$, and the underlying attribute-based signature scheme $\mathsf{ABS.Sign}(TPK, APK, SK_{\mathcal{A}}, m, \Upsilon')$ is called to get a signature ϕ. We return the time-t signature $\sigma = (t, \phi)$.
– $\mathsf{RABS.Ver}(TPK, APK, m, \Upsilon, \sigma)$: Using time t from the signature $\sigma = (t,\phi)$, compute $\Upsilon' = T(\Upsilon, t)$ as in the Sign function, and then verify signature ϕ using $\mathsf{ABS.Ver}(TPK, APK, m, \Upsilon', \phi)$.
– $\mathsf{RABS.ReIssue}(\tau, \psi, pid, TPK, ASK, \mathcal{A} = \{(a_1,\iota_1),\ldots,(a_u,\iota_u)\})$: Performs the same checks on $\mathcal{A}$ as in $\mathsf{RABS.AttrGen}$, and further verifies that the underlying attribute set $(a_1, a_2, \ldots, a_u)$ has not changed from the previous time period. If $\mathcal{A}$ verifies, create validity attribute $v = (\mathsf{validity}, [t_s,t_e])$ and call $\mathsf{ABS.Extend}(\tau, \psi, ASK, \{v\})$, returning the resulting (Δ, ψ').
– $\mathsf{RABS.Extend}(\tau, \psi, pid, TPK, ASK, \mathcal{A}' = \{(a'_1,\iota'_1),\ldots,(a'_u,\iota'_u)\})$: Recover the current attribute set $\mathcal{A}$ for set pid from ψ, and let U be the union of validity intervals for all validity attributes that have been issued with this set (note that U may not be a contiguous interval). Next, check that all ι'_i designate the same standard validity interval $[t'_s,t'_e]$ and that $[t'_s,t'_e] \subseteq U$, returning an error if this is not true. Finally, if $[t'_s,t'_e] = U$ we call $\mathsf{ABS.Extend}(\tau, \psi, pid, TPK, ASK, \mathcal{A}')$, returning the resulting (Δ, ψ'); otherwise, $[t'_s,t'_e]$ is a proper subset of U, and this operation generate an entirely new signing key by pairing each attribute of $\mathcal{A}$ with $[t'_s,t'_e]$ and calling $\mathsf{RABS.AttrGen}(\tau, TPK, ASK, \mathcal{A} \cup \mathcal{A}')$ to get (SK, ψ), giving $(\langle \textsc{New}, SK\rangle, \psi)$ as the result of the Extend operation.

The extra check in Extend is subtle, but important: Since we can reissue an attribute set for a new time interval by just issuing a new validity attribute, there may be older validity attributes that are carried along with the attribute set. If we did not make this test, then a new attribute added by using Extend would be in the same attribute set as an older validity attribute, allowing a dishonest user to create a signature with a time that pre-dates when they were authorized to use the new attributes. Note that in all cases the underlying attribute set is extended, even if the validity interval for the set is being restricted in this special case.

Theorem 2. *Given a secure non-time based ABS scheme, Implementation 2 is a secure RABS scheme.*

Proof. Like in the proof of Theorem 1, given a RABS instance in which security properties are considered, we consider the "underlying" ABS operations that are performed in any execution of RABS operations, either in regular use or in answering oracle requests in a security game. Use of a RABS predicate Υ in a Sign or Verify is converted using the same transformation described for Implementation 2, so the resulting ABS instance uses predicate $\Upsilon' = T(\Upsilon, t)$. The underlying ABS instance has an enlarged set of standard attributes, $\mathbb{A} \cup \mathbb{V}$, but any semantic meaning about "time" can be ignored when considered in the ABS scheme. Consider a signature $\sigma = (t, \phi)$ made by the RABS.Sign operation using a valid (i.e., issued and with appropriate time/validity attributes for time t) set of attributes with $\Upsilon(\mathcal{A}) = 1$. By Lemma 1, the issued attributes in the ABS scheme are a set $\mathcal{A} \cup \mathcal{V}$ such that $\Upsilon'(\mathcal{A} \cup \mathcal{V}) = 1$, and the *Correctness* of the RABS scheme follows directly from the Correctness of the underlying ABS scheme.

Next, consider any two RABS signatures made with differing valid attribute sets for a common predicate Υ. Since $\Upsilon' = T(\Upsilon, t)$ is deterministic, these signatures correspond to two signatures in the underlying ABS scheme with the same predicate Υ', and so are identically distributed because of the Perfect Privacy property of the underlying ABS scheme. *Perfect Privacy* of the RABS scheme follows.

Finally, consider the Existential Unforgeability game for this RABS implementation. Any forged signature that wins this game must correspond to a forged signature in the underlying ABS game, with valid ABS attributes (this follows from Lemma 1 again, but using the opposite direction of the "if and only if" from its use for the Perfect Privacy property). Since the RABS oracles and the translation from RABS to ABS do not require any secrets, we can turn any sequence of adversary operations and queries in RABS into a similarly successful forgery sequence against the underlying ABS, where the running time of the ABS adversary is at most polynomial in the running time of the RABS adversary. Since the underlying ABS scheme is secure, no probabilistic polynomial time adversary can win this game with non-negligible probability, and hence no probabilistic polynomial time adversary can win the RABS game with non-negligible probability, and hence this RABS implementation is *Existentially Unforgeable.* □

4.4 Implementation 3: Grouped Validity Intervals

In Implementation 1, each attribute was given a validity interval that was independent of all other attributes, while in Implementation 2, all attributes shared a common validity interval. In Implementation 3 we take the middle ground: we partition the attribute set $\mathbb{A}$ into b buckets of size $p = |\mathbb{A}|/b$ so that all attributes in the same bucket share a common validity interval. While Implementation 2 supported efficient reissue of all attributes, excluding (revoking) a single attribute on reissue would require reissuing the entire set. While Implementation 3 is considerably more complex, it supports efficient full-set reissue by issuing a single validity attribute, and partial-set reissue with $O(\log b)$ overhead.

In order to understand this implementation, we first review some mathematical and cryptographic definitions related to bilinear pairings.

Groups with Bilinear Pairings. The techniques we develop are based on cryptographic systems built over groups with bilinear pairings with certain properties, so we first define these concepts. Let $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ be prime order cyclic groups of order $p \in \mathbb{Z}$. Let g and h be generators of $\mathbb{G}$ and $\mathbb{H}$ respectively. Then $e : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ is a bilinear pairing if $e(g, h)$ is a generator of $\mathbb{G}_T$, and if $\forall\, a, b$, it holds true that $e(g^a, h^b) = e(g, h)^{ab}$. The following three cryptographic assumptions are commonly used in cryptographic applications of groups with bilinear pairings:

1. q-Strong Diffie Hellman assumption [1]: The q-strong Diffie Hellman (q-SDH) assumption holds in $(\mathbb{G}, \mathbb{H})$ if, given the elements $(g, g^x, g^{x^2}, \cdots, g^{x^q}, h, h^x) \in \mathbb{G}^{q+1} \times \mathbb{H}^2$, for any $x \in \mathbb{Z}_p, g \in \mathbb{G}, h \in \mathbb{H}$, it is computationally infeasible to compute any pair of the form $\left(c, g^{\frac{1}{x+c}}\right) \in \mathbb{Z}_p \times \mathbb{G}$.
2. Symmetric External Diffie Hellman assumption [6]: The symmetric external Diffie Hellman (SXDH) assumption holds in $(\mathbb{G}, \mathbb{H})$ if the standard Decisional Diffie Hellman assumption holds simultaneously in ($\mathbb{G}$ and in $\mathbb{H}$).
3. Decision Linear assumption [2]: Let $\mathbb{G} = \mathbb{H}$. The Decision Linear (DLIN) assumption holds in $\mathbb{G}$ if, given the elements $(g^x, g^y, g^{rx}, g^{sy}, g^t) \in \mathbb{G}^5$, for any $x, y, r, s \in \mathbb{Z}_p$, it is computationally infeasible to determine if $t = r + s$ or if t is random in $\mathbb{Z}_p$.

Boneh-Boyen Signatures. The first building-block we need is a signature scheme over bilinear groups, due to Boneh and Boyen [1]. The operations of the Boneh-Boyen digital signature scheme are defined as follows:

1. $(VK, SK) \leftarrow \mathsf{DS.KeyGen}(p, g)$: Choose $b, c, d \in \mathbb{Z}_p$, compute $B = g^b$, $C = g^c$, $D = g^d$. Set $VK = (B, C, D) \in \mathbb{G}^3$, and set $SK = (b, c, d) \in \mathbb{Z}_p^3$.
2. $\sigma \leftarrow \mathsf{DS.Sign}(SK, h, m \in \mathbb{Z}_p)$: Choose an $r \in \mathbb{Z}_p$, compute $\sigma = \left(h^{\frac{1}{b+cm+dr}}, r\right) \in \mathbb{H} \times \mathbb{Z}_p$. Output σ.
3. $\{0, 1\} \leftarrow \mathsf{DS.Ver}(VK, m, \sigma)$. Let $S = h^{\frac{1}{b+cm+dr}}$. Output 1 if $e(BC^m D^r, S) = e(g, h)$, output 0 otherwise.

The Boneh-Boyen signature scheme is strongly unforgeable under the q-SDH assumption.

Non-Interactive Witness Indistinguishable Proofs of Knowledge. The next tool we need is that of a Non-Interactive Witness Indistinguishable (NIWI) Proof of Knowledge, and the specific implementation of Groth and Sahai [6].

Definition 3. *(NIWI Proofs of Knowledge)*

1. $\mathsf{NIWI.Setup}$*: Outputs a reference string* crs*.*
2. $\mathsf{NIWI.Prove}$*: On input* (crs, Φ, x)*, where* Φ *is a boolean formula and* $\Phi(x) = 1$*, outputs a proof* π*.*
3. $\mathsf{NIWI.Verify}$*: On input* (crs, Φ, π) *outputs a boolean value.*
4. $\mathsf{NIWI.SimSetup}$*: Outputs a simulated reference string* crs *and trapdoor* ψ*.*
5. $\mathsf{NIWI.Extract}$*: On input* (crs, ψ, Φ, π)*, outputs a witness* x*.*

Definition 4. *(NIWI Security Properties)*

1. Completeness*:* $\mathsf{NIWI.Verify}(crs, \Phi, \mathsf{NIWI.Prove}(crs, \Phi, x)) = 1$ *if* $\Phi(x) = 1$*.*
2. Witness Indistinguishability*: If* $\Phi(x_1) = 1$ *and* $\Phi(x_2) = 1$*, then the distributions of* $\mathsf{NIWI.Prove}(crs, \Phi, x_1)$ *and* $\mathsf{NIWI.Prove}(crs, \Phi, x_2)$ *are identical.*
3. Proof of Knowledge: *The* crs *output by* $\mathsf{NIWI.Setup}$ *is indistinguishable from the* crs *output by* $\mathsf{NIWI.SimSetup}$*. Furthermore, if* $(crs, \psi) \leftarrow \mathsf{NIWI.SimSetup}$ *and* $\mathsf{NIWI.Verify}(crs, \Phi, \pi) = 1$*, then* $\mathsf{NIWI.Extract}(crs, \psi, \Phi, \pi)$ *outputs a valid witness for* Φ *with overwhelming probability.*

Groth and Sahai provide two NIWI instantiations, with security based on either the DLIN or SXDH assumption [6]. While there are other differences in these two schemes, either will work in our RABS application.

RABS Implementation 3. To refer to an attribute $a \in \mathbb{A}$ that was issued as part a specific attribute set, say set *pid*, we will use a subscript like a_{pid}. For example, an `Employee` attribute issued in set 512 could be written as $\texttt{Employee}_{512}$. When we explicitly specify the *pid* for an issued attribute, like a_{pid}, we call this as an "identified attribute."

We define a special type of attribute, called a *link attribute*, which will serve as a bridge between two issued attribute sets. Like any other attribute, a link attribute is issued as part of a particular issued attribute set, say set *pid*, but it also specifies the issued attribute set to which it is linking, which we will denote *opid* (for "other *pid*"). The name of such an attribute is written as $\mathsf{link}\sharp opid$, and once issued we can identify a specific identified link attribute as $\mathsf{link}\sharp opid_{pid}$. A link attribute indicates that issued attributes from sets *pid* and *opid* can be used together in making a signature, as if they had been issued in the same set.

Consider a situation in which a user has been issued two different attribute sets — specifying as identified attributes, the user might have an $\mathsf{Employee}_{592}$ attribute from the first set, and $\mathsf{ExtendedHours}_{811}$ from the second set. Since these are parts of different sets (with different *pid*s), they couldn't normally be

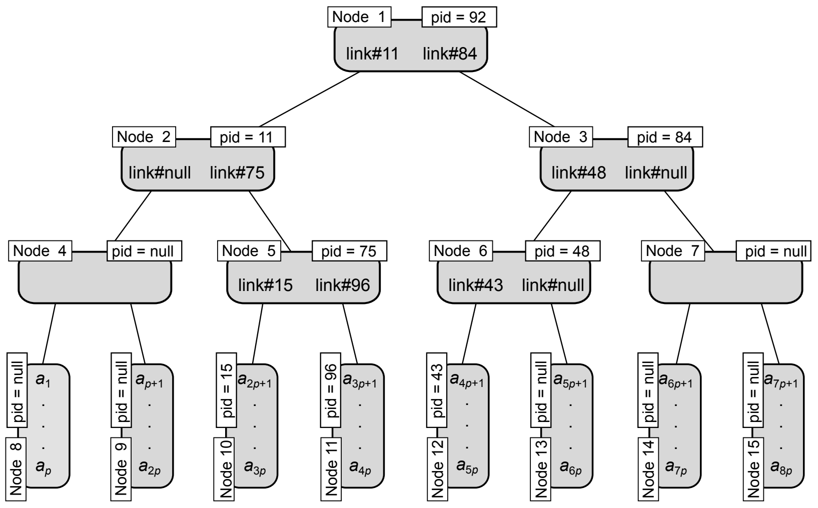

Fig. 1. Example Attribute Tree

used together to satisfy a predicate Υ. However, if the AIA issues a $\mathsf{link}\sharp 811$ attribute to the user as part of the "592" attribute set, then supplying this link attribute when making a signature would enable the signature to be made using both $\mathsf{Employee}_{592}$ and $\mathsf{ExtendedHours}_{811}$ (or any attributes from these two sets). If there are only two sets to link together, as in this simple example, it is easy to modify the RABS.Sign and Verify functions to enforce the necessary properties. However, we would like to support more than just two-set scenarios, so we next describe how to build a tree of attributes so that an arbitrary number of attribute sets can be linked together and will define the sign and verify operations with respect to these attribute trees.

Attribute Trees. As described above, we partition the set $\mathbb{A}$ into b buckets of $p = |\mathbb{A}|/b$ attributes each. While we can generalize to any sizes, in this section we assume that b is powers of two. Consider a complete binary tree with b leaves, where each node in the tree can have an independently issued attribute set. The leaves correspond to the b attribute buckets, and each internal node of the tree corresponds to an attribute set that can contain only link attributes. We use the standard 1-based numbering of nodes in a complete binary tree (as used in heaps) to identify positions in the tree, so "Node 1" is the root of the tree.

The AIA maintains such a tree for each user, with that user's current issued attributes. An example is shown in Figure 1, where leaf nodes 10, 11, and 12 have issued attribute sets (with *pid*s as shown in the figure), but other leaf nodes do not (indicated with $pid = null$). The AIA maintains a mapping between tree nodes and *pid*s, and we can write $pid[u, node]$ to refer to the current *pid* for user u and node *node*, or just $pid[node]$ when the user is known. For example, in Figure 1 the AIA's mapping contains $pid[1] = 92$, $pid[5] = 75$, $pid[9] = null$, etc.

As in the previous implementations, issued attribute sets contain validity attributes, and a validity attribute stored in any node of the tree overrides validity intervals specified in its children and by transitivity all of its descendants. This enables the reissue of the entire attribute set by simply adding a new validity attribute to the root node. We define two functions that are useful in describing how attribute issue and reissue work.

Given a set of attributes $\mathcal{A}$, define the *bucket set* $bs(\mathcal{A})$ to be the set of leaf nodes containing attributes from $\mathcal{A}$, and define the *ancestor set* $as(\mathcal{A})$ to be set of all proper ancestors of the bucket set. In the example in Figure 2,

$$bs(\{a_{2p+1}, a_{3p+1}, a_{4p+1}\}) = \{\mathsf{Node\ 10}, \mathsf{Node\ 11}, \mathsf{Node\ 12}\}, \text{ and}$$

$$as(\{a_{2p+1}, a_{3p+1}, a_{4p+1}\}) = \{\mathsf{Node\ 1}, \mathsf{Node\ 2}, \mathsf{Node\ 3}, \mathsf{Node\ 5}, \mathsf{Node\ 6}\}.$$

Whenever the AIA issues an attribute set $\mathcal{A}$, it issues sets for nodes in $bs(\mathcal{A})$ as well as the internal nodes $as(\mathcal{A})$. The ancestor set always forms a connected subset of the attribute tree, and therefore if link attributes are interpreted as meaning "attribute sets *pid* and *opid* can be used together," then proving possession of the link attributes for the nodes in the ancestor set proves that all associated attribute sets can be used together to satisfy the predicate. After a set is issued (as represented by a connected subset of an attribute set), whenever a leaf node needs to be reissued by, for example, removing an attribute from a bucket, then we issue a new set for that leaf and link the new set in by issuing issue new sets for the internal nodes on the path from that leaf to the root.

The key management operations, $\mathsf{AttrGen}$, $\mathsf{ReIssue}$, and Extend can all be handled uniformly using the $\mathsf{UpdateTree}$ function shown in Figure 2. These functions are given access to the state ψ that the AIA saves regarding the keys in this attribute tree. For simplicity of notation, we define the following functions which extract parts of the information represented by ψ: $\mathsf{Set}(node, \psi)$ denotes the set of attributes associated with any node in the tree, $\mathsf{State}(node, \psi)$ denotes the *node*-specific state, and $\mathsf{Params}(\psi)$ denote the τ parameters for the attribute tree's user. We also use $\mathsf{IntervalIntersection}(\mathcal{A})$ to denote the intersection of all intervals represented in a time-specific attribute set $\mathcal{A}$. $\mathsf{AttrGen}$ calls $\mathsf{UpdateTree}$ with a null state ψ since this does not modify or build on an existing tree, whereas the two update functions ($\mathsf{ReIssue}$ and Extend) provide the saved state associated with the root of the tree.

Sign and Verify Operations. The previous description of attribute trees could be applied to any attribute-based signature scheme, but to handle link attributes in the $\mathsf{RABS.Sign}$ and $\mathsf{RABS.Ver}$ operations we use a specific implementation based on Boneh-Boyen signatures [1]. Note that Maji *et al.* [10] present this with an additional layer of abstraction for "credential bundles," but we present directly in terms of Boneh-Boyen signatures. In particular, the AIA's keypair (APK, ASK) is a Boneh-Boyen signature keypair, and the "secret key" for attribute a in issued attribute set *pid* is the digital signature $\mathsf{DS.Sign}(ASK, pid\|a)$. Since only the AIA could have created such a signature, proving possession of this signature is the same as proving that this attribute was issued by the AIA. To issue a link attribute $\mathsf{link}\sharp opid_{pid}$ the AIA computes $\mathsf{DS.Sign}(ASK, pid\|\mathsf{link}\sharp opid)$.

```
$\mathsf{ProcessLeaf}(node, pid, \psi, ASK, \mathcal{A})$
    if $\mathsf{IntervalIntersection}(\mathcal{A}) = \emptyset$ then return $\langle \textsc{Error}, \bot, \bot, \bot \rangle$
    $\mathcal{A}' \leftarrow \mathsf{Set}(node, \psi)$ // The pre-update attribute set
    if $\mathcal{A}'$ is empty or $|\mathcal{A}' - \mathcal{A}| > 0$ then
        $(SK, pid', \psi') \leftarrow \mathsf{RABS1.AttrGen}(\mathsf{Params}(\psi), TPK, ASK, \mathcal{A})$
        return $\langle \textsc{Issue}, \mathsf{link}\sharp pid', \{(node, \langle \textsc{new}, SK \rangle)\}, \{(node, \psi')\} \rangle$
    else
        $v \leftarrow (\mathsf{validity}, \mathsf{IntervalIntersection}(\mathcal{A}))$ // new validity attribute
        $(\Delta, \psi') \leftarrow \mathsf{RABS1.Extend}(\mathsf{Params}(\psi), \mathsf{State}(node, \psi), pid, TPK, ASK,$
                        $(\mathcal{A} - \mathcal{A}') \cup \{v\})$
        return $\langle \textsc{Extend}, v, \{(node, \Delta)\}, \{(node, \psi')\} \rangle$

$\mathsf{UpdateTree}(node, \psi, pid, ASK, \mathcal{A})$
    if $node$ is a leaf node then return $\mathsf{ProcessLeaf}(node, \psi, pid, ASK, \mathcal{A})$
    $\mathcal{A}' \leftarrow \mathsf{Set}(node, \psi)$ // The pre-update attribute set
    if $\mathcal{A} = \mathcal{A}'$ and $\mathsf{IntervalIntersection}(\mathcal{A}) \neq \emptyset$ then
        $v \leftarrow (\mathsf{validity}, \mathsf{IntervalIntersection}(\mathcal{A}))$ // new validity attribute
        $(\Delta, \psi') \leftarrow \mathsf{RABS1.Extend}(\mathsf{Params}(\psi), \mathsf{State}(node, \psi), pid, TPK, ASK, \{v\})$
        return $\langle \textsc{Extend}, v, \{(node, \Delta)\}, \{(node, \psi')\} \rangle$
    else
        $action \leftarrow \textsc{Extend}$
        $V \leftarrow \{\}$
        for $c \in \mathsf{children}(node, \psi)$ do
            $\langle label, attr, SK', \psi' \rangle \leftarrow \mathsf{UpdateTree}(c, \psi, pid[c], ASK, \mathsf{PartitionAttr}(c, \mathcal{A}))$
            if $label = \textsc{Error}$ then return $\langle \textsc{Error}, \bot, \bot, \bot \rangle$
            if $label = \textsc{Issue}$ then
                $action \leftarrow \textsc{Issue}$
                Replace link attr in $\mathcal{A}'$ with $attr$
            $V \leftarrow V \cup \{attr\}$
            $SK_{new} \leftarrow SK_{new} \cup SK'$
            $\psi_{new} \leftarrow \psi_{new} \cup \psi'$

        $v \leftarrow (\mathsf{validity}, \mathsf{IntervalIntersection}(V))$ // new validity attribute
        if $action = \textsc{Issue}$ then
            $(SK, pid', \psi') \leftarrow \mathsf{RABS1.AttrGen}(\mathsf{Params}(\psi), TPK, ASK, \mathcal{A}' \cup \{v\})$
            return $\langle \textsc{Issue}, \mathsf{link}\sharp pid', SK_{new} \cup \{(node, \langle \textsc{new}, SK \rangle)\}, \psi_{new} \cup \{(node, \psi')\} \rangle$
        else
            $(\Delta', \psi') \leftarrow \mathsf{RABS1.Extend}(\mathsf{Params}(\psi), \mathsf{State}(node, \psi), pid, TPK, ASK, \{v\})$
            return $\langle \textsc{Extend}, v, SK_{new} \cup \{(node, \Delta')\}, \psi_{new} \cup \{(node, \psi')\} \rangle$
```

Fig. 2. Key-management functions. RABS1 is a single-set RABS scheme.

In the example in Figure 2, the right child link attribute pictured for Node 5 would be the signature $\mathsf{DS.Sign}(ASK, 75\|\mathsf{link}\sharp 96)$.

A signature in this attribute-based signature then is a non-interactive witness indistinguishable (NIWI) proof showing that the signer knows signatures corresponding to a set of attributes that satisfy the predicate Υ. For a predicate Υ that depends on ℓ attributes, $a_1, \ldots, a_\ell$, a signer may not have been issued all attributes, so we use notation $\bot$ to denote a "null signature" — a value that is in the form of a proper signature, but does not verify. The signer then creates a list of signatures $\sigma_1, \ldots, \sigma_\ell$ which may be either actual secret keys (Boneh-Boyen signatures) or the value $\bot$. Any non-$\bot$ signature provided should be a verifiable signature, and these should correspond to a set of attributes that satisfy Υ. Therefore, in Maji *et al.*'s signature scheme (without attribute trees), the signature is a NIWI proof of (modified slightly from Maji *et al.*):

$$\begin{aligned}\exists\, pid, \sigma_1, \cdots, \sigma_\ell : &\left(\bigwedge_{i=1,\ldots,\ell} (\sigma_i = \bot) \vee \mathsf{DS.Ver}\left(APK, pid \| a_i, \sigma_i\right) = 1 \right) \\ &\wedge \Upsilon\left(\{a_i \mid \sigma_i \neq \bot\}\right) = 1. \end{aligned} \tag{2}$$

Modifying this technique to use attribute trees, first note that a predicate Υ that refers to attribute set $\mathbb{A}_\Upsilon$ will reference a subset of the attribute tree with a total of $nn = |bs(\mathbb{A}_\Upsilon)| + |as(\mathbb{A}_\Upsilon)|$ nodes. Therefore, Υ references nn separately issued attribute sets and hence there are nn distinct pids to account for in the NIWI statement. In this subset of the attribute tree there are $nn - 1$ link attributes, and since each link attribute and each base attribute is issued as a signature from the AIA, there are a total of $ns = |\mathbb{A}_\Upsilon| + nn - 1$ signatures.

To construct the NIWI statement, we order the ns signatures into a sequence so that $\sigma_1, \ldots, \sigma_{|\mathbb{A}_\Upsilon|}$ are signatures for base attributes and $\sigma_{|\mathbb{A}_\Upsilon|+1}, \ldots, \sigma_{ns}$ are signatures for link attributes. We order the nn nodes of the attribute subtree arbitrarily so that $pid_1, \ldots, pid_{nn}$ are the pids of the sets issued at all relevant nodes. We define a map $n : [1, \ldots, ns] \to [1, \ldots, nn]$ so that $n(i)$ gives the node containing signature/attribute σ_i, so issued set for signature σ_i is $pid[n(i)]$. Since every node except the root node is linked from its parent by a link attribute, given as a signature, we define $p : [1, \ldots, ns] \to [0, \ldots, ns]$ so that $p(i)$ is the parent link to the node containing σ_i; for the root node r we define $p(r) = 0$. Finally, we let $lnk : [|\mathbb{A}_\Upsilon| + 1, \ldots, ns] \to [1, \ldots, nn]$ be such that if σ_i represents a link attribute then $lnk(i)$ is the child node that this link attribute connects to. To simplify notation, we will use $\ell(i)$ to denote node i's label, which is either $pid[n(i)] \| a_i$ (for a leaf node) or $pid[n(i)] \| \mathsf{link} \sharp pid[lnk(i)]$ (for an internal node). The $\mathsf{RABS.Sign}$ and $\mathsf{RABS.Verify}$ operations then create and verify a NIWI proof of the following predicate:

$$\begin{aligned}&\exists pid_1, \cdots, pid_{nn}, \sigma_1, \cdots, \sigma_{ns} : \\ &\bigwedge_{i=1,\ldots,ns} \left((\sigma_i = \bot) \vee \left[(\mathsf{DS.Ver}(APK, \ell(i), \sigma_i) = 1) \wedge \left(p(i) = 0 \vee \sigma_{p(i)} \neq \bot\right)\right]\right) \\ &\quad \wedge \quad \Upsilon\left(\{a_i \mid 1 \leq i \leq n \quad \wedge \quad \sigma_i \neq \bot\}\right) = 1\end{aligned}$$

Just like in (2), a user does not have to have signatures for all attributes in $\mathbb{A}_\Upsilon$ to satisfy this statement, but if the user *does* supply a signature for a non-root attribute then it must also provide a signature for the link from its parent. This ensures that the attributes used by the signer (which must satisfy Υ by the last clause) are all connected by link attributes indicating that they can all be used together even though issued in different attribute sets.

Theorem 3. *Under the q-SDH and either DLIN or SXDH assumptions, Implementation 3 is a secure RABS scheme.*

Proof. We focus here on how the necessary predicates can be efficiently encoded for use in the Groth-Sahai proof system. Perfect Privacy and Existential Unforgeability properties follow from the security properties of the NIWI proof properties. In the Groth-Sahai proof system, the prover first creates commitments to the witnesses and proves that the committed values satisfy equations constructed over bilinear maps.

In Equation 4.4, the prover generates bitwise commitments to $pid_1, \cdots, pid_{nn}$, and $r \in \mathbb{Z}_p$ from the underlying Boneh Boyen signature scheme. The prover then sets $\forall\, i \in [1..nn]\,, pid_i = \Sigma_j\, pid_{i_j} \cdot 2^j$ and $r = \Sigma_j\, r_j \cdot 2^j$. the prover can then generate bitwise commitments to $pid_1 \cdots, pid_{nn}$ and r. Let $\mathbb{G}$ and $\mathbb{H}$ be cyclic groups of prime order p, and let $g \in \mathbb{G}$, $h \in \mathbb{H}$ be generators of their groups. The prover generates commitments for $g^{pid_{i_j}}, g^{r_i}, h^{pid_{i_j}}, h^{r_i}$, over $\mathbb{G}$ and $\mathbb{H}$, and proves the following pairing equations for all values of i, where $\langle val \rangle$ denotes a commitment to value val:

$$
\begin{gathered}
e(\langle g^{pid_{i_j}} \rangle, h) = e(g, \langle h^{pid_{i_j}} \rangle); \\
e(\langle g^{r_i} \rangle, h) = e(g, \langle h^{r_i} \rangle); \\
e(\langle g^{pid_{i_j}} \rangle, \langle h^{pid_{i_j}} \rangle) = e(\langle g^{pid_{i_j}} \rangle, h); \\
e(\langle g^{r_i} \rangle, \langle h^{r_i} \rangle) = e(\langle g^{r_i} \rangle, h)
\end{gathered}
$$

Next, to show how to encode in the Groth-Sahai NIWI proof system, the prover needs to construct pairing equations over $\mathbb{G}$ and $\mathbb{H}$ for the statements $\mathsf{DS.Ver}(APK, pid[n(i)] \| a_i, \sigma_i) = 1$ and $\mathsf{DS.ver}(APK, pid[n(i)] \| \mathsf{link} \sharp pid[lnk(i)], \sigma_i) = 1$. With respect to the Boneh-Boyen signature scheme as given above, $e(BC^{pid[n(i)] \| a_i} D^r, S) = e(g, h)$ is logically equivalent to the expression $\mathsf{DS.Ver}(APK, pid[n(i)] \| a_i, \sigma_i) = 1$, where $\sigma = (S, r)$. And the equation $e(BC^{pid[n(i)] \| \mathsf{link} \sharp pid[lnk(i)]} D^r, S) = e(g, h)$ is logically equivalent to the expression $\mathsf{DS.Ver}(APK, pid[n(i)] \| \mathsf{link} \sharp pid[lnk(i)], \sigma_i) = 1$, where $\sigma = (S, r)$. The prover then constructs and proves the following pairing equations:

$$
\begin{gathered}
e(\langle D^r \rangle, h) = \prod_j e(D^{2^j}, \langle h^{r_j} \rangle); \\
e(\langle C^{pid_i} \rangle, h) = \prod_j e(C^{2^j}, \langle h^{pid_{i_j}} \rangle); \\
e(g, h) = e(BC^{a \cdot 2^{|pid_i|}}, \langle S \rangle)\; e(\langle C^{pid_i} \rangle, \langle S \rangle)\; e(\langle D^r \rangle, \langle S \rangle)
\end{gathered}
$$

Finally, Groth-Sahai proofs can be instantiated using Waters' signature scheme [16] under the Decision Linear or Symmetric External Diffie Hellman assumptions. This completes the proof. □

Efficiency: Since the tree is a complete binary tree with b leaves (i.e., buckets), there are $\log_2 b$ link nodes on the path from any attribute to the root. Therefore, $|as(\mathbb{A}_\Upsilon)| \leq |\mathbb{A}_\Upsilon| \log_2 b$ (with equality when $|\mathbb{A}_\Upsilon| = 1$), and since $|bs(\mathbb{A}_\Upsilon)| \leq |\mathbb{A}_\Upsilon|$ we have $nn \leq |\mathbb{A}_\Upsilon|(1 + \log_2 b)$ and $ns \leq |\mathbb{A}_\Upsilon|(2 + \log_2 b)$. Therefore, compared to the single-set NIWI proof, this implementation adds an overhead factor of $O(\log b)$. Smaller numbers of buckets require less overhead, but this savings must be balanced against the increased cost of issuing new attribute sets for leaves (which can be substantial if there are large numbers of attributes in each bucket).

5 Counter-based Revocation (Revocation-on-Demand)

In this section, we illustrate the use of counter-based time (as defined in Section 2.1) to create an implemention in which we can revoke attributes on demand. Recall that in counter-based time, time consists of a vector of integers $\langle t_1, t_2, \cdots, t_k\rangle$ where each t_i corresponds to a monotonic counter. Each independently specifiable validity interval requires its own counter in this vector, so the vector dimension varies based on which RABS implementation we are using. For example, in Implementation 1, each attribute has an independently-specified validity interval, so we require $|\mathbb{A}|$ counters, giving times that are $|\mathbb{A}|$-dimensional vectors. For large sets of attributes, this is very inefficient, but consider the other extreme: in Implementation 2, all attributes share the same validity interval, so in that case we only need a single counter, and a time t is just a single value for this counter. The counter-based technique is most interesting in Implementation 3, where for b buckets we use b counters, one for each bucket.

To provide trusted counter values, one could potentially use a "counter service" to maintain the counters, which can either be provided by the attribute-issuing authority, or by a separate dedicated server. The counter service can be implemented in software by the attribute-issuing authority, or the attribute-issuing authority (alternatively a separate, dedicated server) could be equipped with trusted hardware that maintains the counters. One way of generating a large number of counters using limited trusted hardware is the virtual monotonic counter construction proposed by Sarmenta *et al.* [14]. We use the same notation for the attribute set as in previous sections: $\mathcal{A} = \{(a_1, \iota_1), \cdots, (a_u, \iota_u)\}$, where each ι_i is of the form $\iota_i = [t_{s_i}, t_{e_i}]$, but in this case the time values are vectors of counter values. Intervals are restricted in this setting, where specifying an interval $[t_s, t_e]$ means giving vectors $t_s = \langle t_{s,1}, t_{s,2}, \cdots, t_{s,k}\rangle$ and $t_e = \langle t_{e,1}, t_{e,2}, \cdots, t_{e,k}\rangle$ where, for each $i = 1, \ldots, k$, either

(a) $t_{s,i} = t_{e,i}$ or
(b) $t_{s,i} = 0$ and $t_{e,i} = \infty$.

Condition (a) gives a specific value for a counter, while condition (b) is basically a "don't care" condition for that monotonic counter.

Example: Consider a instance of Implementation 3, with $|\mathbb{A}| = 64$ and attributes numbered $0, \ldots, 63$, an attribute tree with $b = 4$ buckets so that attribute a_i would be mapped to bucket number $\lfloor i/16 \rfloor$ for $i = 0, \ldots, 63$, and we assign a counter ctr_i to bucket i for $i = 0, \ldots, 3$. When making a signature for a predicate with $\mathbb{A}_\Upsilon = \{a_3, a_{12}, a_{18}, a_{55}\}$, this involves ctr_0 (for a_3 and a_{12}), ctr_1 (for a_{18}) and ctr_3 (for a_{55}). If $\mathsf{ctr}_0 = 32$, $\mathsf{ctr}_1 = 49$, $\mathsf{ctr}_2 = 44$, and $\mathsf{ctr}_3 = 12$, then an interval saying that all the relevant attributes for Υ are at the "current time" could be $[\langle 32, 49, 0, 12\rangle, \langle 32, 49, \infty, 12\rangle]$. Since we have a "don't care" condition for ctr_2, signatures of the form $\sigma = (t, \phi)$ with $t = \langle 32, 49, 45, 12\rangle$ or $t = \langle 32, 49, 46, 12\rangle$ or even $t = \langle 32, 49, 324, 12\rangle$ would all be in the required time interval.

Instantly revoking attributes is now easy: if user Alice has been issued both a_3 and a_{12}, and the attribute-issuing authority decides to revoke a_{12} from Alice, then ctr_0 is incremented. This invalidates attributes $a_0, \ldots, a_{15}$ held by all users, including a_3 held by Alice, so these must all be reissued. However, since this is only one fourth of the entire univers of attributes, this is more efficient than having to reissue all attributes to all users. Revoking an attributed tied to counter ctr_2 would result in incrementing ctr_2, but since this is a "don't care" condition in the interval for Υ, signatures both before and after ctr_2 is incremented are equally valid.

6 Conclusion

In this paper we have initiated a careful study of incorporating time intervals into attribute-signatures, so that attributes can be given a finite lifespan when they are issued. This allows for attribute revocation either at pre-defined time instances (in our clock-based techniques) or on demand (in our counter-based technique). This is preliminary work in this direction, and there are many open questions related to supporting different models of time as well as improving efficiency. One possible direction of future work is to explore revoking attributes while using non-monotone span programs [18], as this would help represent a richer range of predicates. From an efficiency standpoint, it would be useful to explore revocability in the setting of attribute-based signature construction techniques that avoid the use of non-interactive witness indistinguishable proofs.

References

1. Boneh, D., Boyen, X.: Short signatures without random oracles. In: EUROCRYPT. pp. 56–73 (2004)
2. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: CRYPTO. pp. 41–55 (2004)
3. Boneh, D., Sahai, A., Waters, B.: Functional encryption: a new vision for public-key cryptography. Commun. ACM 55(11), 56–64 (2012)
4. Escala, A., Herranz, J., Morillo, P.: Revocable attribute-based signatures with adaptive security in the standard model. In: AFRICACRYPT. pp. 224–241 (2011)
5. Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B.: Attribute-based encryption for circuits from multilinear maps. In: CRYPTO. pp. 479–499 (2013)
6. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: EUROCRYPT. pp. 415–432 (2008)
7. Herranz, J., Laguillaumie, F., Libert, B., Ràfols, C.: Short attribute-based signatures for threshold predicates. In: CT-RSA'12. pp. 51–67 (2012)
8. Hohenberger, S., Waters, B.: Online/offline attribute-based encryption. In: PKC. pp. 293–310 (2014)
9. Li, J., Au, M.H., Susilo, W., Xie, D., Ren, K.: Attribute-based signature and its applications. In: ASIACCS. pp. 60–69 (2010)
10. Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures. In: CT-RSA. pp. 376–392 (2011)
11. Okamoto, T., Takashima, K.: Efficient attribute-based signatures for non-monotone predicates in the standard model. In: Public Key Cryptography. pp. 35–52 (2011)

12. Okamoto, T., Takashima, K.: Decentralized attribute-based signatures. In: Public Key Cryptography. pp. 125–142 (2013)
13. Rouselakis, Y., Waters, B.: Practical constructions and new proof methods for large universe attribute-based encryption. In: ACM CCS. pp. 463–474 (2013)
14. Sarmenta, L.F.G., van Dijk, M., O'Donnell, C.W., Rhodes, J., Devadas, S.: Virtual monotonic counters and count-limited objects using a TPM without a trusted OS. In: ACS CCS, STC. pp. 27–42 (2006)
15. Shahandashti, S.F., Safavi-Naini, R.: Threshold attribute-based signatures and their application to anonymous credential systems. In: AFRICACRYPT. pp. 198–216 (2009)
16. Waters, B.: Efficient identity-based encryption without random oracles. In: EUROCRYPT. pp. 114–127 (2005)
17. Waters, B.: Functional encryption: Origins and recent developments. In: PKC. pp. 51–54 (2013)
18. Yamada, S., Attrapadung, N., Hanaoka, G., Kunihiro, N.: A framework and compact constructions for non-monotonic attribute-based encryption. In: PKC. pp. 275–292 (2014)