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, USA
srtate@uncg.edu
[2] Department of Computer Science, SUNY Poly, Utica, NY 13502, USA
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.* [8] introduced *attribute-based signatures* as a primitive that allows users to sign messages anonymously using a combination of their attributes. The parties involved in attribute-based signatures are a signature trustee (ST), an attribute-issuing authority (AIA), and potentially many signers and verifiers.

This material is based upon work supported by the National Science Foundation under Grant No. 0915735.

© IFIP International Federation for Information Processing 2015
P. Samarati (Ed.): DBSec 2015, LNCS 9149, pp. 153–169, 2015.
DOI: 10.1007/978-3-319-20810-7_10

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.* [8], the basic scheme uses attributes that do not have any time restrictions on validity – once an attribute is issued, it good forever (or at least as long as the global public verification key is valid). Maji *et al.* [8] 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.* [8], but they don't give any specifics on how attribute sets can incorporate signing key revocation or attribute set revocation. Escala *et al.* [3] 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.* [7], Shahandashti and Safavi-Naini [12], and Herranz *et al.* [5] present attribute-based signature schemes, but do not deal with attribute and key revocation and expiry. Okamoto and Takashima [9,10] 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* [2,4,6,11,14], 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.* [8]. 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.* [8]. 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 $\Upsilon(\{a_i \,|\, i = 1, \ldots, n\}) = 1$ 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 $t =$ `2014-March`. These times form a totally ordered set, and larger intervals can be specified such as [`2014-March`, `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

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

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.

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:*

- 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*.*
- 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.*
- RABS.ASetup$(TPK, 1^\lambda) \rightarrow (APK, ASK)$*: Run by the attribute-issuing authority (AIA) to generate a keypair* (APK, ASK)*.*
- 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 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*.*
- 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.*
- RABS.Ver$(TPK, APK, m, \Upsilon, \sigma) \rightarrow v \in \{$*"accept","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* $\Upsilon(\mathcal{A}) = 1$*.*
- 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.*
- 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* $\mathcal{A} \cup \mathcal{A}'$*.*
- 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.* [8] — 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* 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.* [8] will work, but Implemention 3 has more strict requirements, as described in Sect. 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 $(\texttt{Employee}, [\texttt{2014-Jan-06}, \texttt{2014-Jan-12}])$ would be named `Employee-2014-Jan-06-2014-Jan-12`.

Using the notion of standard validity intervals from Sect. 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 Sect. 2.1, if $a(i) = \texttt{Employee}$ 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 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 $t \in T$:

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

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 $\Upsilon = (\mathbf{M}, a)$, where $\mathbf{M}$ is an $\ell \times k$ matrix, to create a time-specific monotone access program $\Upsilon' = (\mathbf{M}', a')$ 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 Υ and the time t, we can denote this as $\Upsilon' = T(\Upsilon, t)$.

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

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

Proof. Let $\Upsilon = (\mathbf{M}, a)$ and $\Upsilon' = (\mathbf{M}', a')$, 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 $\Upsilon(\mathcal{A}) = 1$, 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 $\Upsilon'(\mathcal{A} \cup \mathcal{V}) = 1$. Since $\Upsilon(\mathcal{A}) = 1$, 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, $\Upsilon(\mathcal{A}') = 1$.

For the other direction of the "if and only if," let $\mathcal{A}'$ be a set of attributes such that $\Upsilon'(\mathcal{A}') = 1$, 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 $\Upsilon(\mathcal{A}) = 1$. 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. □

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, $\mathsf{Register}$, $\mathsf{ASetup}, \mathsf{Update}$) carry over without modification. $\mathsf{AttrGen}$ and $\mathsf{Reissue}$ require a check to make sure that all specified validity attributes are the same, and Sign, and Ver require modifications based on transforming $\Upsilon' = T(\Upsilon, t)$ and incorporating the time t into the signature. For space reasons, definitions for the basic functions is left to the full version of this paper [13]. The one tricky function is Extend, which is defined and explained below:

– $\mathsf{RABS.Extend}(\tau, \psi, pid, TPK, ASK, \mathcal{A}' = \{(a'_1, \iota'_1), \ldots, (a'_u, \iota'_u)\}) \rightarrow (\Delta, \psi')$: 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. A proof for the following theorem is in the full version of this paper [13].

Theorem 2. *Given a non-interactive witness indistinguishable proof of knowledge, and a secure credential bundle scheme, Implementation 2 is a secure revocable attribute-based signature scheme.*

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 involving a single validity attribute, and partial with with $O(\log b)$ overhead.

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.

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 Fig. 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 Fig. 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 Fig. 2,

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

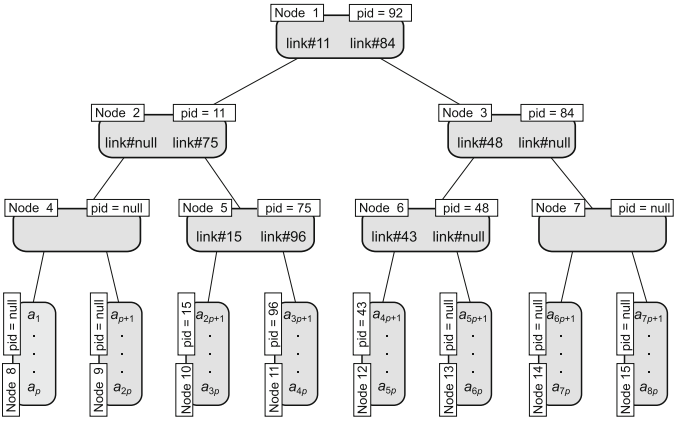

Fig. 1. Example Attribute Tree

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

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, AttrGen, RelIssue, and Extend can all be handled uniformly using the UpdateTree function shown in Fig. 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 $\mathcal{A}$. AttrGen calls UpdateTree with a null state ψ since this does not modify or build on an existing tree, whereas the two update functions (RelIssue 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 RABS.Sign and RABS.Ver operations we use the specific credential-bundle implementation of attribute-based signatures given by Maji *et al.* [8], where the credential bundles are implemented using Boneh-Boyen signatures [1]. In particular,

```
ProcessLeaf(node, pid, ψ, ASK, A)
    if IntervalIntersection(A) = ∅ then return ⟨Error, ⊥, ⊥, ⊥⟩
    A' ← Set(node, ψ) // The pre-update attribute set
    if A' is empty or |A' − A| > 0 then
        (SK, pid', ψ') ← RABS1.AttrGen(Params(ψ), TPK, ASK, A)
        return ⟨Issue, link♯pid', {(node, ⟨new, SK⟩)}, {(node, ψ')}⟩
    else
        v ← (validity, IntervalIntersection(A)) // new validity attribute
        (Δ, ψ') ← RABS1.Extend(Params(ψ), State(node, ψ), pid, TPK, ASK,
                               (A − A') ∪ {v})
        return ⟨Extend, v, {(node, Δ)}, {(node, ψ')}⟩

UpdateTree(node, ψ, pid, ASK, A)
    if node is a leaf node then return ProcessLeaf(node, ψ, pid, ASK, A)
    A' ← Set(node, ψ) // The pre-update attribute set
    if A = A' and IntervalIntersection(A) ≠ ∅ then
        v ← (validity, IntervalIntersection(A)) // new validity attribute
        (Δ, ψ') ← RABS1.Extend(Params(ψ), State(node, ψ), pid, TPK, ASK, {v})
        return ⟨Extend, v, {(node, Δ)}, {(node, ψ')}⟩
    else
        action ← Extend
        V ← {}
        for c ∈ children(node, ψ) do
            ⟨label, attr, SK', ψ'⟩ ← UpdateTree(c, ψ, pid[c], ASK, PartitionAttr(c, A))
            if label =Error then return ⟨Error, ⊥, ⊥, ⊥⟩
            if label =Issue then
                action ← Issue
                Replace link attr in A' with attr
            V ← V ∪ {attr}
            SK_new ← SK_new ∪ SK'
            ψ_new ← ψ_new ∪ ψ'

        v ← (validity, IntervalIntersection(V)) // new validity attribute
        if action = Issue then
            (SK, pid', ψ') ← RABS1.AttrGen(Params(ψ), TPK, ASK, A' ∪ {v})
            return ⟨Issue, link♯pid', SK_new ∪ {(node, ⟨new, SK⟩)}, ψ_new ∪ {(node, ψ')}⟩
        else
            (Δ', ψ') ← RABS1.Extend(Params(ψ), State(node, ψ), pid, TPK, ASK, {v})
            return ⟨Extend, v, SK_new ∪ {(node, Δ')}, ψ_new ∪ {(node, ψ')}⟩
```

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

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 $\mathsf{DS.Sign}(ASK, pid \| \mathsf{link}\sharp opid)$. In the example in Fig. 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 $\perp$ 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 $\perp$. Any non-$\perp$ 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.*):

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

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 *pid*s 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 *pid*s 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 RABS.Sign and RABS.Verify operations then create and verify a NIWI proof of the following predicate:

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

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. A proof of the following theorem can be found in the full version of this paper [13].

Theorem 3. *Given a NIWI proof of knowledge and a secure credential bundle scheme, Implementation 3 is a secure RABS scheme.*

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 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 [15], 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: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
2. Boneh, D., Sahai, A., Waters, B.: Functional encryption: a new vision for public-key cryptography. Commun. ACM **55**(11), 56–64 (2012)
3. Escala, A., Herranz, J., Morillo, P.: Revocable attribute-based signatures with adaptive security in the standard model. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 224–241. Springer, Heidelberg (2011)
4. Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B.: Attribute-based encryption for circuits from multilinear maps. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 479–499. Springer, Heidelberg (2013)
5. Herranz, J., Laguillaumie, F., Libert, B., Ràfols, C.: Short attribute-based signatures for threshold predicates. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 51–67. Springer, Heidelberg (2012)
6. Hohenberger, S., Waters, B.: Online/offline attribute-based encryption. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 293–310. Springer, Heidelberg (2014)
7. Li, J., Au, M.H., Susilo, W., Xie, D., Ren, K.: Attribute-based signature and its applications. In: ASIACCS, pp. 60–69 (2010)
8. Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 376–392. Springer, Heidelberg (2011)
9. Okamoto, T., Takashima, K.: Efficient attribute-based signatures for non-monotone predicates in the standard model. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 35–52. Springer, Heidelberg (2011)
10. Okamoto, T., Takashima, K.: Decentralized attribute-based signatures. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 125–142. Springer, Heidelberg (2013)
11. Rouselakis, Y., Waters, B.: Practical constructions and new proof methods for large universe attribute-based encryption. In: ACM CCS, pp. 463–474 (2013)
12. Shahandashti, S.F., Safavi-Naini, R.: Threshold attribute-based signatures and their application to anonymous credential systems. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 198–216. Springer, Heidelberg (2009)

13. Tate, S.R., Vishwanathan, R.: Expiration and revocation of keys for attribute-based signatures. Cryptology ePrint Archive, report 2015/xxx (2015). http://eprint.iacr.org/2015/xxx
14. Waters, B.: Functional encryption: origins and recent developments. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 51–54. Springer, Heidelberg (2013)
15. Yamada, S., Attrapadung, N., Hanaoka, G., Kunihiro, N.: A framework and compact constructions for non-monotonic attribute-based encryption. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 275–292. Springer, Heidelberg (2014)