HGAA: An Architecture to Support Hierarchical Group and Attribute-Based Access Control

Daniel Servos
Western University
London, Ontario
dservos5@uwo.ca

Sylvia L. Osborn
Western University
London, Ontario
sylvia@csd.uwo.ca

ABSTRACT

Attribute-Based Access Control (ABAC), a promising alternative to traditional models of access control, has gained significant attention in recent academic literature. This attention has lead to the creation of a number of ABAC models including our previous contribution, Hierarchical Group and Attribute-Based Access Control (HGABAC). However, to date few complete solutions exist that provide both an ABAC model and architecture that could be implemented in real life scenarios.

This work aims to advance progress towards a complete ABAC solution by introducing Hierarchical Group Attribute Architecture (HGAA), an architecture to support HGABAC and close the gap between a model and real world implementation. In addition to HGAA we also present an attribute certificate specification that enables users to provide proof of attribute ownership in a pseudonymous and off-line manner, as well as an update to the Hierarchical Group Policy Language (HGPL) to support our namespace for uniquely identifying attributes across disparate security domains.

Details of our HGAA implementation are given and a preliminary analysis of its performance is discussed as well as directions for future work.

CCS CONCEPTS

• **Security and privacy → Access control**; *Authentication*; *Security protocols*; *Domain-specific security and privacy architectures*;

KEYWORDS

ABAC, Attribute-Based Access Control, HGABAC, Hierarchical Group and Attribute-Based Access Control, Architecture, Access Control, Attribute Certificate, Attribute Authority, HGAA, Hierarchical Group Attribute Architecture

ACM Reference Format:
Daniel Servos and Sylvia L. Osborn. 2018. HGAA: An Architecture to Support Hierarchical Group and Attribute-Based Access Control. In *ABAC'18: 3rd ACM Workshop on Attribute-Based Access Control, March 19–21, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3180457.3180459

ABAC'18, March 19–21, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5633-6/18/03…$15.00
https://doi.org/10.1145/3180457.3180459

1 INTRODUCTION

Attribute-Based Access Control (ABAC) is an emerging form of access control that bases access control decisions on the attributes of users, objects and the environment rather than the identity of users or the roles/clearances assigned to them. While the beginnings of ABAC in academic literature can be seen as early as 15 years ago[19], ABAC has only recently gained significant attention in the past half decade[15]. This newfound interest has resulted in the creation of numerous ABAC models[5, 10, 12, 13], however, none to date have gained acceptance as a unified standard or provided a complete view of how they might be implemented in practice. Real-world implementation details and results are still needed and existing models largely lack architectural specifications required for actual use and empirical study.

In our pervious work[13], we introduced Hierarchical Group and Attribute-Based Access Control (HGABAC), a model of Attribute-Based Access Control that incorporates hierarchical user and object groups to ease administration and increase policy flexibility. Since publication, extensions to and expansion of the original HGABAC model have been explored by ourselves and others, including introduction of an adminstative model ($GURA_G$)[7], the creation of an authorization architecture for a restricted HGABAC model (rHGABAC)[3] and preliminary work towards incorporating delegation concepts[14]. However, none of these works provide a complete architecture to facilitate real-world implementation and use of HGABAC in a distributed environment.

Questions like "who assigns the attributes?", "how are attributes shared with each party?", "how does the user provide proof of attribute ownership?", "where and how are policies evaluated?", "how will the model scale in real-world use?", etc. remain unanswered. This paper attempts to answer these questions and more in regards to HGABAC through the creation of an attribute-based architecture, entitled Hierarchical Group Attribute Architecture (HGAA). HGAA enables the use of HGABAC in distributed environments by formalizing and providing the following key components:

- **Attribute Authority**: A service for managing, storing and providing user attributes in the form of attribute certificates that are used to authenticate with a user service provider.
- **Attribute Certificate**: A cryptographically secured certificate that details attributes a user has activated for a given session as well as revocation and delegation rules under which the certificate was issued.
- **HGABAC Namespace**: A URI-based namespace for uniquely identifying HGABAC elements (attributes, users, objects, etc.) across disparate security domains and authorities.
- **Policy Authority**: A service which manages and evaluates HGABAC policies on behalf of a user service provider.

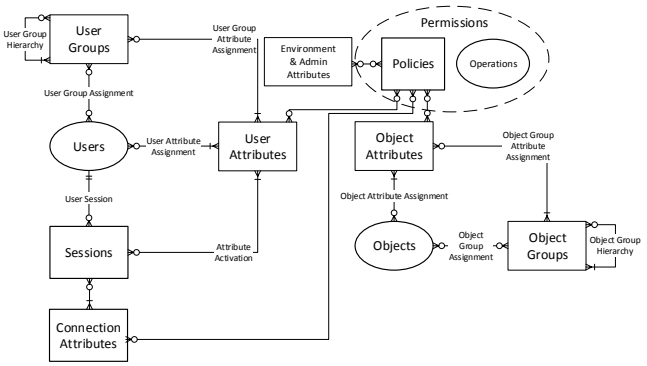

Figure 1: HGABAC components and relations using Crow's Foot Notation to denote cardinality of relationships. Primitive components are shown in ovals.

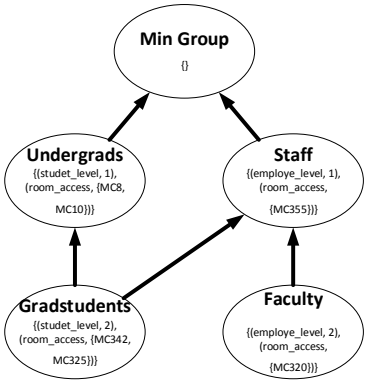

Figure 2: Example user group hierarchy represented as a graph. The large bold text denotes the group's name, beneath which the set of directly assigned attributes is shown.

Listing 1: Example HGABAC HGPLv1 Policy Permission Pairs

```
P1 = (user.age >= 18 AND object.title = "Adult␣Only␣Book", read)
P2 = (user.id = object.author, write)
P3 = (user.role IN {"doctor", "intern", "staff"} AND
      user.id != object.patient, read)
P4 = (object.type = "program" AND object.required_certifications
      SUBSET user.certifications, run)
```

- **User Service Provider**: A provider of services to end users that have access restricted on the basis of one or more HGABAC policies.

These service components are implemented as JSON-based web services and performance results are given to demonstrate the scalability of the architecture in terms of number of attributes and policy complexity. A detailed attribute certificate format is specified that includes support for future delegation and revocation extensions as discussed in [14].

The remainder of this paper is laid out as follows; Section 2 briefly introduces HGABAC and gives background information, Section 3 discusses related work and it's applicability to HGAA, Section 4 introduces the overall architecture and details each component, Section 5 details our HGAA implementation and gives preliminary performance results and finally Section 6 presents our concluding remarks and considers directions for future work.

2 HGABAC BACKGROUND

HGABAC[13] provided a formal model of ABAC that introduced group based hierarchical representations of object and user attributes that was not available in other models at the time. In HGABAC, attributes are assigned both directly to access control entities and indirectly assigned through user and object attribute groups (these relations are shown in Figure 1). Attribute groups help simplify administration of ABAC systems by allowing administrators to create user or object groups whose membership indirectly assigns sets of attribute/value pairs to its members. These groups are hierarchical and inherit attribute/value pairs from their parent groups allowing for more flexible policy representation when combined with the HGABAC policy language.

The group hierarchy is represented as a directed acyclic graph with each group a vertex and each edge a parent/child relation between the groups such that the edge is directed to the parent. All possible paths in the graph have the constraint that they must eventually lead to a special *min_group* that has no parents and no assigned attributes. Child groups inherit the attributes of their parent groups such that the child group's "effective" set of attributes (i.e. the set of attributes both directly assigned to the group and inherited from other groups) consists of the union of all parent groups' "effective" attributes with the attributes directly assigned to the child. An example user group graph is shown in Figure 2, in which directly assigned attributes are shown under each group name. In this example, the effective set of attributes for the *Gradstudents* group would be $\{\{employee\_level, \{1\}\}, \{student\_level, \{1, 2\}\}, \{room\_access, \{MC8, MC10, MC355, MC342, MC325\}\}$ as the *employee_level* attribute is inherited from the *Staff* group and values of the other attributes are merged with the values from the parent groups (*Staff* and *Undergrads*). In the case of the *Faculty* group, the effective set of attributes would be $\{\{employee\_level, \{1, 2\}\}, \{room\_access, \{MC355, MC320\}\}$, only inheriting attributes from the *Staff* group.

In addition to the core HGABAC model, an attribute-based policy language (HGPLv1) was created to support policy creation and evaluation. HGPLv1 represents policies as C style boolean statements that may evaluate to *TRUE*, *FALSE* or *UNDEFINED*. A resulting evaluation of *TRUE* implies that access should be granted, *FALSE* that it should be denied and *UNDEFINED* if the policy can not be properly evaluated at the current time (equivalent to a result of *FALSE* for access control decision purposes). Policies are associated with a set of operations that they grant if satisfied.

Listing 1 presents a number of example policies that are possible in HGPLv1. Policy P1 states that any user with an age of 18 or older can read the book with the title "Adult Only Book". Policy P2 allows a user to write to any object they are an author of. Policy P3 limits access to read a medical record to users who have the role

Listing 2: XACML rule to only allow access between 9AM and 5PM.

```
<Rule RuleId="TimeRule" Effect="Permit">
  <xacml3:Description>Allow if time between 9AM and 5PM</xacml3:Description>
  <xacml3:Target/>
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
        <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
        <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
    </Apply>
  </Condition>
</Rule>
```

Listing 3: HGABC Policy to only allow access between 9AM and 5PM.

```
env.time_of_day_hour >= 9 AND env.time_of_day_hour <= 17
```

doctor, intern or staff but only if they are not listed as a patient in that record. And finally, policy P4 specifies that a user can run a program if they have the required certifications listed in the program's certifications attribute.

It has been shown[13] that HGPLv1 and HGABAC are capable of emulating MAC, DAC and hierarchical RBAC (though not separation of duties) and that their attribute groups result in less complex (in terms of the number of assignments and relations between access control entities) representations than standard (non-hierarchical) ABAC models under a number of hypothetical use cases.

3 RELATED WORK

A number of generic access control frameworks and architectures exist that could conceivably be used to support HGABAC. The most notable of these are the Security Assertion Markup Language (SAML)[9], the eXtensible Access Control Markup Language (XACML)[1] and the AAA Authorization Framework[18]. SAML provides an XML-based standard for exchanging authentication and authorization information commonly used for single sign-on (SSO). XACML provides a XML-based standard for representing and sharing access control policies and an accompanying architecture that follows the AAA Authorization Framework which describes an authorization framework as a combination of the following distributed policy modules:

- **Policy Retrieval Point (PRP)**: Point at which a policy is retrieved from a Policy Repository.
- **Policy Decision Point (PDP)**: Point where a policy from a PRP is evaluated based on information from one or more PIPs.
- **Policy Enforcement Point (PEP)**: Point at which the policy is enforced (i.e. the point at which a user is denied or granted access to a service based on the result of the PDP).
- **Policy Information Points (PIP)**: Point at which information needed to evaluate a policy is retrieved. In the case of ABAC, this is normally attributes and their values.
- **Policy Administration Point (PAP)**: Point at which policies are administered and/or created[1].

While it may be possible to create a HGABAC architecture-based system solely on these standards (and this is a possible direction for future work mentioned in Section 6), such a solution would face a number of issues and shortcomings addressed by our HGAA architecture:

- **Off-Line PIPs**: In the most common use of the XACML architecture, a PDP requests the policy information required to evaluate a policy from a PIP after it receives a request from a PEP (presumably triggered by a user attempting to access a service). However, this approach is problematic if PIPs are unavailable (e.g. a user's home domain may not have a PIP for user attributes that is available continuously or may wish to only assign attributes off-line) or such requests introduced excessive overhead (e.g. if a large number of PIPs need to be contacted to collect a full set of a user's attributes). HGAA provides a solution in which no outside PIPs need be contacted if an attribute certificate is available.
- **Public Key Infrastructure (PKI) Overhead**: X.509 Attribute Certificates[4] and SAML require the use of X.509 PKI which may be unavailable or overkill in many cases. Additionally, requiring attribute certificates to be accompanied by the holder's Public Key Certificate can weaken anonymity, a key feature of ABAC. HGAA overcomes this by using a simplified public key based trust system between Policy and Attribute Authorities and specifying an attribute certificate format that does not require X.509 Public Key Certificates and allows for pseudonymity.
- **Future Support for Delegation**: A future goal of HGABAC is to support one or more models of delegation as described in [14]. While XACML does have a delegation profile[11], the style of delegation supported is closer to administration than traditional user-to-user delegation. HGAA provides a number of points upon which future extensions enabling a variety of delegation and revocation models can be built while maintaining backwards compatibility.
- **HGABAC Specific v.s. Generic Architecture**: XACML supports a large range of flexible access control policies that do not necessarily conform to the HGABAC model or policy language.

[1]Not defined in the AAA Authorization Framework, only in XACML

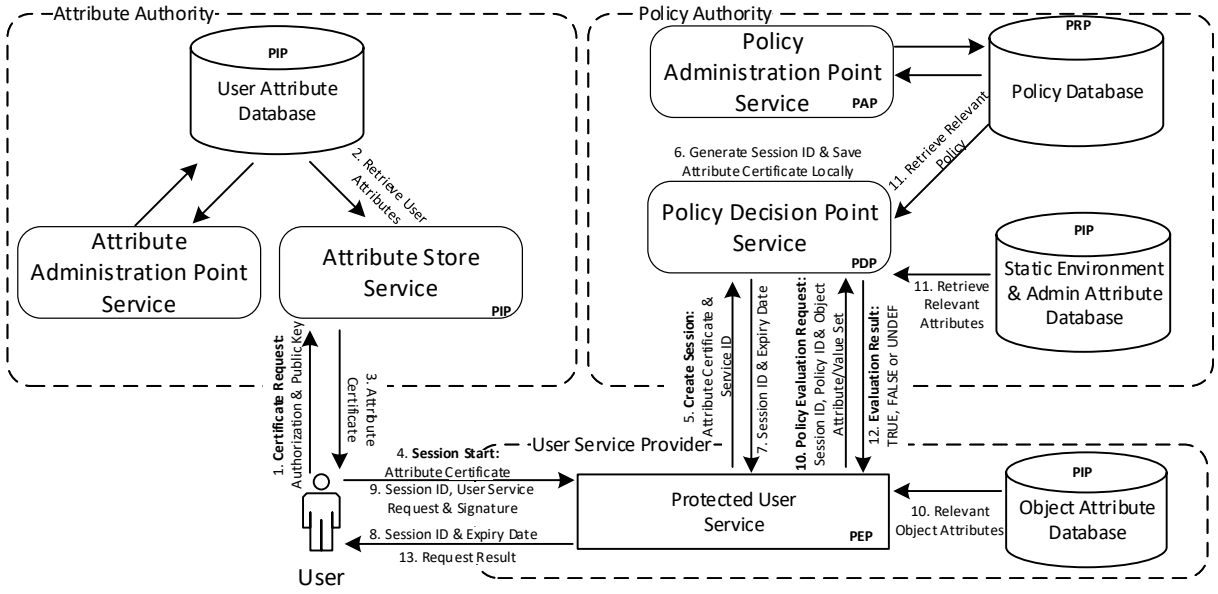

Figure 3: HGAA services and information flow. Numbers indicate order of requests. Dotted lines denote components of a service (i.e. Attribute Store Service is a component/subservice of the Attribute Authority Service).

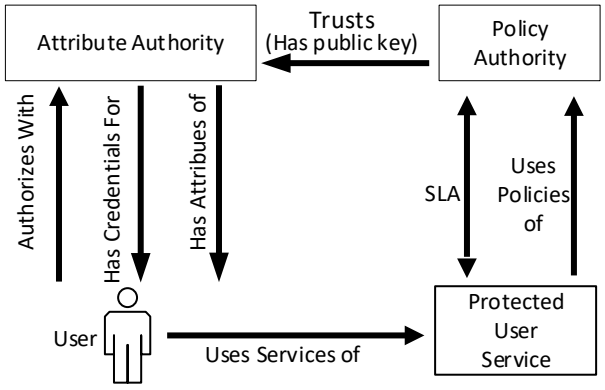

Figure 4: Relationships between HGAA services and users.

Use of XACML would require the creation of a new XACML HGABAC profile and means of translating HGABAC policies into XACML. As HGAA is designed specifically for HGABAC, native HGABAC policies are supported without translation to a secondary policy language and no features are restricted or compromised while ensuring that the HGABAC model is enforced.

- **Lightweight Approach**: XACML and other attribute based architectures provide flexibility and interoperability at the cost of increasing complexity and verbosity of access control policies (for example see the comparison of a XACML and HGABAC policy in Listings 2 & 3) as well as the supporting infrastructure. HGABAC and HGAA take a lightweight approach, aiming to provide a simplified yet powerful policy language that requires minimal infrastructure to support.

In addition to the generic standards mentioned, the work by S. Bhatt et al.[3] towards creating an authorization architecture and implementation of HGABAC utilizing the NIST Policy Machine (PM)[5, 6] is also of note. S. Bhatt et al. introduce a restricted HGABAC model (rHGABAC) formalized as a single-value enumerated policy enabling it to be implemented using a bare minimum version of the PM to take advantage of the PM's existing access control framework. While they were successful in implementing the core features of HGABAC, the restricted model only allowed for simplified policies and lacks all assignment relations from the original HGABAC model. Further, the authorization architecture presented is limited to a single PIP that is combined with the PDP and PAP.

4 ARCHITECTURE

The HGAA architecture is comprised of three core service types; the Attribute Authority service (discussed in Subsection 4.2), the Policy Authority service (discussed in Subsection 4.5) and the services provided by the User Service Provider (referred to as "User Services" and discussed in Subsection 4.4). Using terminology from the AAA Authorization Framework, the Attribute Authority would be analogous to a PIP, the Policy Authority to a combined PDP, PRP and PAP and the user services to a PEP. A simplified (only showing a single instance of each service and one user) representation of this architecture and the information flow between the services is shown in Figure 3.

Many instances of Attribute Authorities, Policy Authorities and User Services are allowed to coexist across diverse security domains. Interoperability between these services and domains is possible so long as a trust relation has been established at a prior time between a given Attribute Authority and a given Policy Authority. From a technical standpoint, this trust relation consists of a Policy

Listing 4: HGABAC URI Based Namespace Grammar

```
Absolute URI:
    hgabac://<authority>[[/<type>]/<element_name>]

Relative URI:
    [/]<type>/<element_name>
    | [/]<element_name>

type:
    user
    | group[/user | /object]
    | attribute[/<att_sub_type>]
    | object[/<obj_sub_type>]
    | session
    | operation
    | permission
    | policy
    | service

att_sub_types:
    user
    | object
    | environment
    | admin
    | connection
    | unknown

obj_sub_types:
    file
    | service
    | hardware
    | program
    | database
    | meta
    | unknown

element_name:
    defined by regex: [a-zA-Z0-9\.\-\_]+

authority:
    <host>[:<port>]

host:
    valid hostname as per RFC 1123

port:
    defined by regex: [1-9][0-9]*
    must be less then 65536.
```

Authority listing an Attribute Authority's public key as trusted (thus accepting attribute certificates issued by this authority). From a practical standpoint, this means that users who are issued an attribute certificate (hereinafter referred to as an "AC" and discussed in more depth in Subsection 4.3) from one domain/organization can access the services of another domain/organization if a trust relation exists between their Attribute and Policy Authorities and the user satisfies the required policies. User Services are able to utilize HGABAC by employing the services provided by a Policy Authority to evaluate user requests (which include an AC issued by a trusted Attribute Authority). These relations between services and users are shown in Figure 4.

The following subsections describe each major HGAA component in more depth including the requests between services shown in Figure 3.

4.1 HGABAC Namespace

As multiple Attribute Authorities may exist and do not directly communicate, attributes from different sources must be uniquely identifiable and conflicts avoided. This issue is not isolated to HGAA but an open ABAC problem that has been identified in recent literature[8, 15]. We offer a partial solution that is similar to the scheme used in XACML, in which each attribute (and all other HGABAC elements) is given a unique URI[2] based on the grammar given in Listing 4. The key difference between our namespace scheme and XACML's is our support for and treatment of relative URIs.

Absolute URIs such as *hgabac://cs1.ca/attribute/user/age* specify a HGABAC element (in this case a user attribute) from a specific authority (in this case *cs1.ca*). A relative URI such as */attribute/user/age* would specify a HGABAC element from any authority (in this case an *age* user attribute from any authority). By omitting other parts of the path, relative URIs can make broader matches. For example, */attribute/role* would match any role attribute regardless of attribute type or issuing authority. Supporting both absolute and relative URIs in this fashion allows policies to be created that reference attributes from a distinct authority (absolute reference) or any attribute of the same name and type (relative reference).

4.2 Attribute Authority

The Attribute Authority provides services to administer the user group hierarchy and user attribute information in a given domain/organization and issue ACs to users within that domain. It is comprised of two subservices and a database (as shown in Figure 3); the Attribute Administration Point Service, the Attribute Store services, and the User Attribute Database. The Attribute Administration Point Service provides administrative operations related to managing and assigning user attributes. The User Attribute Database stores user attribute assignments for users in the Attribute Authority's domain. The Attribute Store Service provides two important functions; first and most importantly it deals with certificate requests and issuing ACs and secondly it maintains a revocation list for all ACs issued in the domain that have been revoked.

The certificate request process proceeds as follows (and is shown as steps 1, 2 and 3 in Figure 3):

(1) The user sends a certificate request to the Attribute Authority containing a list of user attributes they wish to activate for a given session, their public key from a public/private key pair generated solely for this session and their credentials for authenticating with the Attribute Authority. The method of authentication is left as an implementation detail for the Attribute Authority and may be domain specific. As per the HGABAC specification, users may activate a subset of their assigned and inherited user attributes to allow for principle of least privilege and prevent identifying information being included in unneeded attributes.
(2) If the user's credentials are valid, the Attribute Authority requests the activated attributes assigned to the user and their values from the Attribute Database and generates an AC as described in Subsection 4.3. This AC contains the public key provided by the user in step 1, such that the user can provide proof of ownership of the AC using their corresponding private key without providing identifying information.
(3) The generated AC is issued to the user who may now use it to authenticate with User Service Providers including those outside

of the user's home domain without providing any additional credentials. No further communication is required between the user and the Attribute Authority (or the Attribute Authority and any other component of the HGAA) for this session after the certificate has been issued and this process may be completed off-line.

Revocation of ACs may happen in two ways. First, the Attribute Authority may embed revocation rules within the AC that define conditions under which the certificate is valid. For example, a common revocation rule would be to revoke a certificate after a set date/time. The second means of revoking a certificate is through the revocation list maintained by the Attribute Store subservice. AC serial numbers listed in the list are considered to no longer be valid and revoked. Providing this list is optional and requires the Attribute Authority to be on-line and accessible (at least periodically) by Policy Authorities such that revocation lists can be synchronized.

4.3 Attribute Certificate

Attribute Certificates (ACs) allow users to offer proof of their attributes to User Service Providers. This proof comes in the form of a cryptographically signed certificate issued by a trusted Attribute Authority accompanied by the public/private session key pair generated by the user at the start of the session (the public key being embedded in the certificate and the user proving they are in possession of the corresponding private key by signing request).

We loosely base the design of our AC format on X.509 Attribute Certificates[4] but without the need for Public Key Certificates or supporting PKI. The logical format of the certificate is defined using Abstract Syntax Notation One (ASN.1) in Listing 5. Our AC consists of the following 8 sections:

- **ACInformation:** Contains meta information about the AC. Consists of the certificate version number (to facilitate compatibility with future versions), a globally unique serial number[2] and the date and time the certificate was issued.
- **ACIssuer:** Identifying information about the issuer of the AC. Comprised of the public key of the issuer (the attribute authority) including information about the public key algorithm used, a unique identifier for the issuing authority formatted as a HGABAC Namespace URI (e.g. *hgabac://cs1.ca* for the authority *cs1.ca*), and optionally, a common descriptive name for the issuing authority and a URL to the authorities web service if publicly available.
- **ACHolder:** Pseudonymous information about the holder of the AC (i.e. the user the AC was issued to). Contains the public key the user is using for this session including information about the public key algorithm used, a unique identifier for the user formatted as a HGABAC Namespace URI[3] (e.g. *hgabac://cs1.ca/user/u1135* for the authoirty *cs1.ca* and the user *u1135*) and optionally a common descriptive name for the user if anonymity is not desired.
- **Attribute Set:** The set of user attributes and their values that the user wishes to activate. Each attribute in the set must be assigned to the user directly or inherited through group membership as

[2]Only needs to be globally unique for practical purposes and can be based on probability rather than requiring coordination between issuing authorities.

[3]Note that this identifier is a pseudonym for the user and should not contain real identifying information. This identifier can be different or partially randomized for each session.

Listing 5: AC Format Defined in ASN.1

```
AttributeCertificate ::= SEQUENCE {
     information     ACInformation,
     issuer          ACIssuer,
     holder          ACHolder,
     attributes      SEQUENCE OF Attribute,
     revocation      RevocationRules,
     delegation      DelegationRules OPTIONAL,
     extensions      SEQUENCE OF ACExtensions OPTIONAL,
     signature       ACSignature
}

ACInformation ::= SEQUENCE {
     version    ACVersion,
     serial     INTEGER,
     issued     DATE-TIME
}

ACVersion ::= INTEGER { v1(0) }

ACIssuer ::= SEQUENCE {
     issuerPublicKey           BIT STRING,
     issuerKeyAlgorithm        AlgorithmIdentifier,
     issuerUniqueIdentifier    OBJECT IDENTIFIER,
     issuerName                VisibleString OPTIONAL,
     issuerServiceURL          UTF8String OPTIONAL
}

ACHolder ::= SEQUENCE {
     holderPublicKey           BIT STRING,
     holderKeyAlgorithm        AlgorithmIdentifier,
     holderUniqueIdentifier    VisibleString,
     holderName                VisibleString OPTIONAL
}

Attribute ::= SEQUENCE {
     attributeID     OBJECT IDENTIFIER,
     attributeType   OBJECT IDENTIFIER,
     attributeValue ANY DEFINED BY attributeType OPTIONAL,
     attributeName   VisibleString OPTIONAL,
     ...
}

RevocationRules ::= SEQUENCE {
     validAfter                DATE-TIME,
     validBefore               DATE-TIME,
     revocationServiceURL      UTF8String OPTIONAL,
     ...
}

DelegationRules ::= SEQUENCE {
     ...
}

ACExtension ::= SEQUENCE {
     extensionID     OBJECT IDENTIFIER,
     ...
}

ACSignature ::= SEQUENCE {
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING
}

-- As Defined in RFC 5280
AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm       OBJECT IDENTIFIER,
     parameters      ANY DEFINED BY algorithm OPTIONAL
}
```

described in HGABAC. Each attribute contains a unique identifier following the HGABAC Namespace (e.g. *hgabac://cs1.ca/attribute/user/department*), the attribute's type, value and optionally a common descriptive name for the attribute (e.g. "Department Name"). In addtion to the requested user attributes, this set also includes connection attributes that contain meta information about the AC it's self (such as date/time issued, issuer UID, holder UID, group information, etc.) such that these properties can be used in HGABAC policies.

- **ACRevocationRules:** Set of rules under which an AC is valid. For the 1st version of the AC presented in this work, these are limited to valid before and valid after rules. However, space is left for extension and future work. Optionally, a URL to a revocation list web service may be given. Such a service would provide a list of all revoked AC serial numbers issued by the authority.
- **ACDelegationRules:** A place holder for delegation rules to be added in future extensions. It is envisioned that this will allow for Attribute Delegation as described in [14].
- **Extension Set:** A place holder for miscellaneous future extensions. The only requirement given is that extension must have a unique identifier.
- **ACSignature:** A cryptographic signature of all other sections of the attribute certificate using the issuing authority's private key (that corresponds to the public key given in the ACIssuer section). Also included is information about the signature algorithm used. ACs are signed by the attribute authority when issued to users as part of a certificate request and offer proof of the authority trusting that the attributes contained describe the holder of the certificate.

We support both a human readable text-based encoding of an attribute certificate as well as a more efficient and less ambiguous byte encoding detailed in Appendix A. An example of an attribute certificate in the text-based encoding is shown in Listing 6.

Listing 6: Example AC in Text-Based Encoding. Public keys, signature and attribute list abbreviated for length reasons.

```
---- BEGIN ATTRIBUTE CERTIFICATE ----
FORMAT: TEXT
VERSION: 1
==== BEGIN INFORMATION ====
VERSION: 1
SERIAL: 145870283285469230556233521582388196248646048900 3
ISSUED: 1513313064
==== END INFORMATION ====
==== BEGIN ISSUER ====
PUBLIC KEY: LS0tLS1CR ... ZLS0tLS0=
KEY ALGORITHM: RSA[2048]
UID: hgabac://cs1.ca
NAME: CS1.CA Attribute Authority
URL: http://cs1.ca/AttributeAuthority/
==== END ISSUER ====
==== BEGIN HOLDER ====
PUBLIC KEY: LS0tLS1CRU ... VZLS0tLS0=
KEY ALGORITHM: RSA[2048]
UID: hgabac://cs1.ca/user/u1135
NAME: Daniel Servos
==== END HOLDER ====
==== BEGIN ATTRIBUTE SET ====
#### BEGIN ATTRIBUTE: /attribute/user/account_balance ####
ATTRIBUTE ID: /attribute/user/account_balance
ATTRIBUTE TYPE: AttributeType.FLOAT
ATTRIBUTE VALUE: 9999.9999
ATTRIBUTE NAME: account_balance
#### END ATTRIBUTE: /attribute/user/account_balance ####
#### BEGIN ATTRIBUTE: /attribute/user/age ####
ATTRIBUTE ID: /attribute/user/age
ATTRIBUTE TYPE: AttributeType.INT
ATTRIBUTE VALUE: 31
ATTRIBUTE NAME: age
#### END ATTRIBUTE: /attribute/user/age ####
#### BEGIN ATTRIBUTE: /attribute/user/admin ####
ATTRIBUTE ID: /attribute/user/admin
ATTRIBUTE TYPE: AttributeType.BOOL
ATTRIBUTE VALUE: TRUE
ATTRIBUTE NAME: admin
#### END ATTRIBUTE: /attribute/user/admin ####
#### BEGIN ATTRIBUTE: /attribute/user/courses ####
ATTRIBUTE ID: /attribute/user/courses
ATTRIBUTE TYPE: AttributeType.SET
ATTRIBUTE VALUE: CS2211,CS2034,CS1234,CS5678,CS9000
ATTRIBUTE NAME: courses
#### END ATTRIBUTE: /attribute/user/courses ####
#### BEGIN ATTRIBUTE: /attribute/connection/ac_version ####
ATTRIBUTE ID: /attribute/connection/ac_version
ATTRIBUTE TYPE: AttributeType.INT
ATTRIBUTE VALUE: 1
ATTRIBUTE NAME: ac_version
#### END ATTRIBUTE: /attribute/connection/ac_version ####
...many attributes omitted for length reasons...
#### BEGIN ATTRIBUTE: /attribute/connection/aauth_uid ####
ATTRIBUTE ID: /attribute/connection/aauth_uid
ATTRIBUTE TYPE: AttributeType.STRING
ATTRIBUTE VALUE: hgabac://cs1.ca
ATTRIBUTE NAME: aauth_uid
#### END ATTRIBUTE: /attribute/connection/aauth_uid ####
==== END ATTRIBUTE SET ====
==== BEGIN REVOCATION RULES ====
VALID AFTER: 1513313064
VALID BEFORE: 1513316664
URL: http://cs1.ca/AttributeAuthority/revocation_list
==== END REVOCATION RULES ====
==== BEGIN SIGNATURE ====
SIGNATURE ALGORITHM: RSASSA-PKCS1-v1_5:SHA256
SIGNATURE VALUE: j6Zk7zl ... e/eX1nGQ==
==== END SIGNATURE ====
---- END ATTRIBUTE CERTIFICATE ----
```

4.4 User Service Provider

The User Service Provider provides services to end users that they wish to protect using HGABAC and maintain the object group hierarchy and object attribute information relevant to their services. Providers designate the policies under which their services may be accessed but outsource the work of storing and evaluating these policies to the Policy Authority. Requests upon User Services are allowed/denied based on the Policy Authority's evaluation of access policies and the attributes contained in the AC as well as the attributes of objects the service will be accessing and the current value of environment, connection and administrative attributes (as described in HGABAC).

Creation of a session between the user and User Service are handled as follows (shown as steps 4, 5, 7 and 8 in Figure 3):

(4) A user starts a session with a User Service by sending their AC (issued to the user by the Attribute Authority in steps 1-3) to the User Service.
(5) The User Service creates a session for the user by forwarding the user's AC to the Policy Authority in a Create Session request. This request contains the user's AC and the User Service's unique ID as well as authentication information for the User

Service, if authentication between the Policy Authority and User Service is required (depends on implementation).

(7) The Policy Authority responds with session information including a unique session ID and expiry date/time. The User Service saves a copy of the AC and session information until the expiry date/time. For the remainder of the session it is no longer required for the user or User Service to transmit the AC.

(8) The User service responds to the user's session request with the session ID and expiry date/time from the Policy Authority. A session is considered active and valid so long as the AC is not revoked, the session expiry date/time is not past and the user has the session ID and private key corresponding to the public key embedded in the AC. A user may terminate a session by destroying the session ID and/or private key (destroying the private key would terminate all sessions associated with the AC).

Once a session has been established, a user may make requests upon the User Service as follows (shown as steps 9, 10, 12 and 13 in Figure 3):

(9) The user makes a request on the User Service that includes the session ID received in step 8 and signs this request with their private key matching the public key in the AC.

(10) The User Service validates the signature on the request (using the public key in the user's AC) and if valid, sends a Policy Evaluation request to the Policy Authority which includes the session ID, the policy ID associated with the HGABAC policy that must be passed for the request to be allowed and the set of relevant object attributes (and their values).

(12) The Policy Authority responds to the Policy Evaluation request with either *TRUE*, *FALSE*, or *UNDEF* based on the ternary logic used in HGABAC. A *TRUE* response indicates that the policy has been satisfied and the user's request should be allowed. A response of *FALSE* indicates that the policy has not been satisfied and the request should be denied. A *UNDEF* response indicates that the policy could not be evaluated (e.g. an attribute in the policy was not available) and the request should be denied.

(13) Based on the result of the Policy Evaluation request, the User Service either fulfills the user's request and replies with an appropriate response or responds with a message indicating that the request was denied (optionally with more details as to why).

Further requests may be made by the user in a like manner without the need to create a new session so long as the session is not expired or terminated.

4.5 Policy Authority

Policy Authorities store, manage and evaluate HGABAC policies on behalf of User Service Providers. Policies are expressed in the HGABAC Policy Language V2 (HGPLv2), an updated version of the policy language introduced in the original HGABAC work[13]. The grammar for the updated language is given in Listing 7 in Augmented Backus-Naur Form (ABNF). The most notable changes are the use of HGABAC Namespace URIs in place of attribute names and the addition of policy references. Policy references allow policies to reference other policies such that policies can be combined using basic logical operations (*AND*, *OR* and *NOT*). For example, if

Listing 7: HGPLv2 Grammar in ABNF. An update to the HGPL grammar from [13].

```
policy     = policy "OR" term / term
term       = term "AND" exp / exp
exp        = var op var / [ "NOT" ] bool_var
           / [ "NOT" ] "(" policy ")" / [ "NOT" ] policy_id
var        = const / att_id
bool_var   = boolean / att_id
op         = ">" / "<" / "=" / ">=" / "<=" / "!=" / "IN"
           / "SUBSET"
atomic     = int / float / string / "NULL" / boolean
const      = atomic / set
boolean    = "TRUE" / "FALSE" / "UNDEF"
set        = "{" "}" / "{" setval "}"
setval     = atomic / atomic "," setval
int        = [ "-" ] +( DIGIT )
float      = int "." +( DIGIT )
string     = DQUOTE *( %x20-21 / %x23-5B / %x5D-7E
                     / %x5C DQUOTE / %x5C %x5C ) DQUOTE
att_id     = <ATTRIBUTE URI FROM HGABAC NAMESPACE>
policy_id  = <POLICY URI FROM HGABAC NAMESPACE>
```

P1 = "/user/age >= 18 OR /user/parent_consent" and *P2 = "/object/author = /user/id"* then a third policy could be created that references *P1* and *P2*, *P3 = "/policy/P1 AND NOT /policy/P2"* that would be equivalent to *P3 = "(/object/author = /user/id) AND NOT (/object/author = /user/id)"*. Policies are restricted from creating recursive or circular references and references to unavailable policies result in *UNDEF*.

The Policy Authority Service is comprised of two subservices, the Policy Administration Point Service and the Policy Decision Point Service, and two databases, the Policy Database and the Environment & Administrative Attribute Database. The Policy Administration Point Service allows for the creation and management of policies by User Service Providers as well as the management of administrative attributes (as defined in HGABAC). The Policy Database stores HGPLv2 policies available for use by User Services and the Environment & Administrative Attribute Database stores the current values of environment and administrative attributes. The Policy Decision Point Service authenticates user AC and evaluates them against stored policies on behalf of a User Service.

Policy Authorities maintain a list of trusted Attribute Authorities along with their unique ID and public key. AC are considered valid by a Policy Authority if they satisfy the following requirements:

- The issuer of the AC is in the Policy Authority's list of trusted Attribute Authorities.
- The issuer's public key and ID match those recorded by the Policy Authority.
- The issue date of the AC is not in the future and is inside of the valid before and after range given in the revocation rules.
- All revocation rules are met including the current time/date being within the valid before and after range.
- The serial of the AC is not listed in the Attribute Authorities revocation list (if available).
- The version of the AC and all extensions are compatible with the Policy Authority.
- The signature is valid and verifiable with the Attribute Authority's public key.

After a session has been established and the AC authenticated (as described in Subsection 4.4), the User Service evaluates a policy

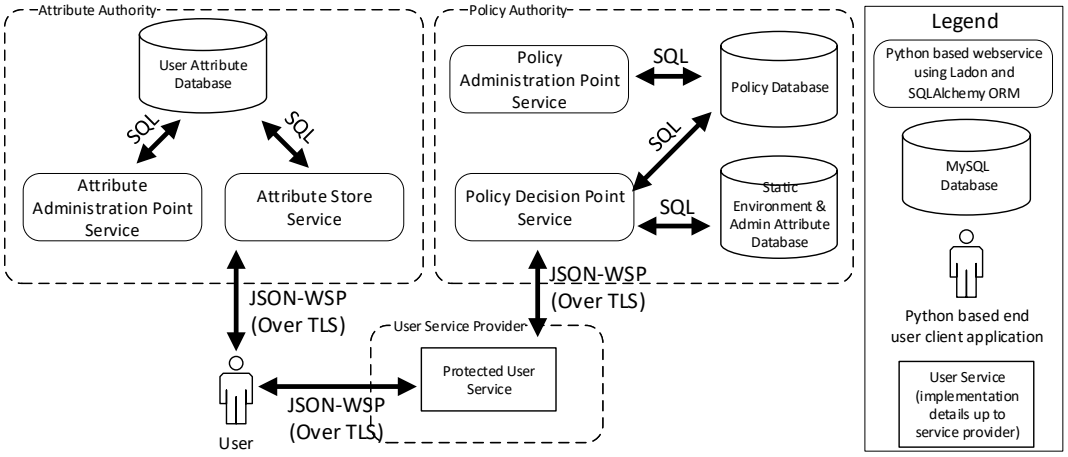

Figure 5: Technology and standards used in the HGAA implementation.

with the Policy Decision Point Service as follows (and shown in steps 10, 11 and 12 of Figure 3):

(10) The User Service sends a Policy Evaluation request that includes the session ID, the policy ID, and the set of object attribute value pairs. The Policy Decision Point Service extracts the user and connection attributes from the user's AC (which it received when the session was created) and checks that the session and AC remain valid and no revocation rules have been triggered.
(11) If the AC remains valid, the decision point requests the policy matching the given policy ID from the Policy Database. If the policy references any other policy, these policies are also requested and combined. The combined policy is analysed and needed environment and administrative attributes are requested from the Environment & Administrative Attribute Database.
(12) The combined policy is evaluated and the result (*TRUE*, *FALSE* or *UNDEF*) is issued to the User Service.

5 IMPLEMENTATION & PRELIMINARY RESULTS

As shown in Figure 5, we implement each HGAA service as a Python-based JSON web service over TLS. Web services are created with the Ladon[16] framework. HGAA databases are implemented using MySQL and the SQLAlchemy[17] Object Relational Mapper (ORM) is used to facilitate communication between services and databases. Administrative services were not implemented at this time as our current focus is on evaluating the performance and scalability of the authentication and authorization features of the architecture.

5.1 Attribute Certificate

We evaluate our AC scheme in terms of size and time required to generate and sign the AC based on the number of user attributes included in the certificate (number of connection attributes remained constant at 35). The result of these comparisons are shown in Figures 6 and 7. We find that the size of the AC grows linearly with the number of user attributes activated at a rate of approximately 36 bytes[4] per attribute (for byte encoding with single value integer attributes) and that the time to generate an AC also grows linearly with the number of user attributes.

5.2 Attribute Authority

The Attribute Store Service of the Attribute Authority was evaluated in terms of request and execution time. The results of this evaluation are shown in Figure 8. Request time is defined as the time it takes a client to generate a Certificate Request, send it to the Attribute Store Service and receive a response (includes network and webserivce overhead), while execution time is defined as the time taken from the point the Attribute Store Service receives a request from a client to the point where the service issues a response (only includes time taken by the service to eventuate the request and generate a response). The number of connection attributes remained constant during testing (at 35).

We find that both the request and execution time increase linearly with the number of user attributes activated and included in the AC. The difference in request and execution time is related to network and web service overhead that we believe can be largely reduced by optimizing our implementation and moving to a different web service framework.

5.3 Policy Authority

An interpreter for the HGPLv2 policy language was implemented using python that utilizes a recursive descent parsing strategy. The interpreter is divided into a number of modules as shown in Figure 12. As an optimization step, the abstract syntax tree (AST) of a given policy is computed and stored in a binary format in the Policy Database when a policy is added or modified. Additionally, an intermediate symbol table, listing the attributes referenced in the policy is computed and stored alongside the AST. Using these precomputed ASTs reduces the time required to fulfill Policy Evaluation requests from User Services and the intermediate symbol table allows the Policy Decision Point Service to request required attributes without reanalysing the policy.

[4]This amount would vary based on a number of factors including the length of the attribute UID, common name, type and number of values.

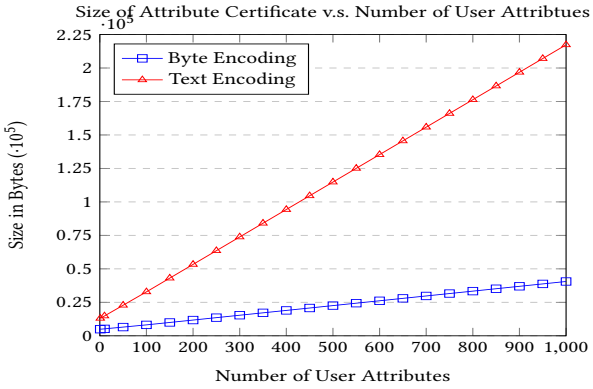

Figure 6: Size of Attribute Certificate v.s. Number of User Attributes

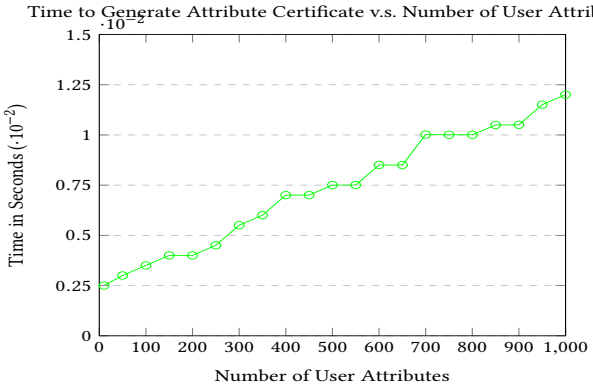

Figure 7: Time to Generate Attribute Certificate v.s. Number of User Attributes

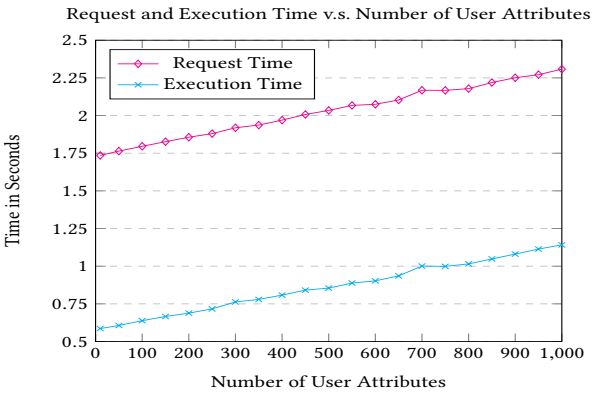

Figure 8: Attribute Store Service Request and Execution Time v.s. Number of User Attributes

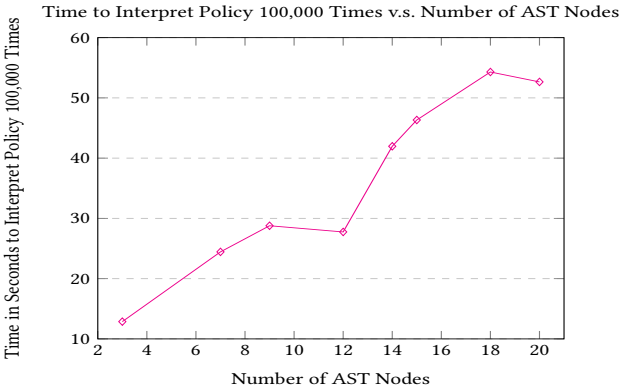

Figure 9: Time to Interpret Policy 100,000 Times v.s. Number of AST Nodes. Policy is evaluated from scratch (not using pre-computed AST).

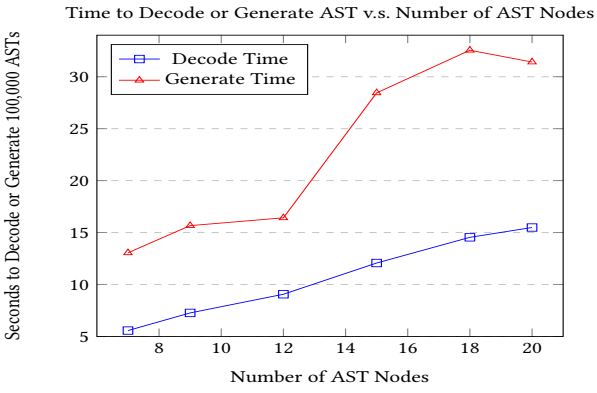

Figure 10: Time to Decode AST Byte Format or Generate AST from Scratch v.s. Number of AST Nodes. Time given is to decode or generate AST 100,000 times.

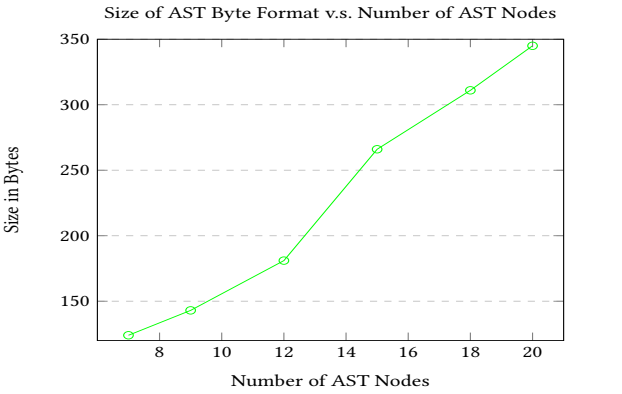

Figure 11: Size of AST Byte Format v.s. Number of AST Nodes.

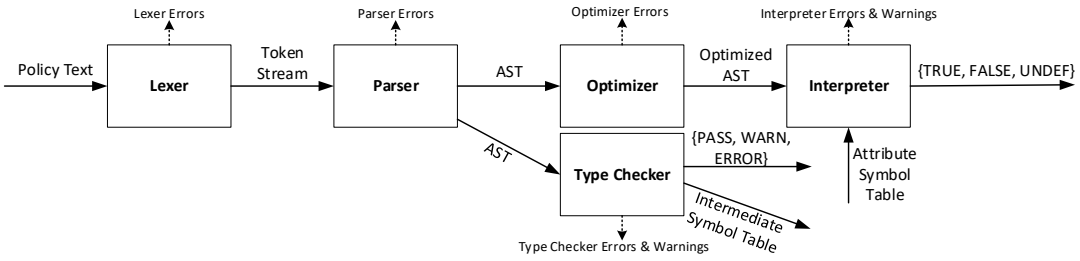

Figure 12: HGPLv2 interpreter modules and information flow.

We evaluated the Policy Authority's performance handling Policy Evaluation requests based on the time required to interpret a policy, the time required to decode a precomputed AST and the size of the precomputed AST. Results of this analysis are shown in Figures 10, 11 and 9. A linear relationship between the number of nodes in a policy's AST and the time required to evaluate the policy was found. Similarly, the time to generate or decode a policy AST also grew linearly with the number of AST nodes as did the size of the AST binary format. AST Nodes are used in place of number of attributes in our analysis as we have found this to be a better measure of a policy's complexity than the number of attributes referenced. However, further testing is needed to evaluate the impact of requesting large number of administrative and environment attributes from the Policy Authority's Attribute Database or the User Service sending a large number of object attributes.

6 CONCLUSIONS & FUTURE WORK

We have introduced the first complete architecture for HGABAC that supports the full model and policy language in a distributed environment. Each architecture component is detailed and the sequence of requests upon each service is described. An attribute certificate specification and encoding is introduced for securely sharing and proving ownership of attributes. The HGPL policy language from [13] is updated to support policy references and our HGABAC Namespace for uniquely identifying attributes (as well as other HGABAC elements) across disparate security domains.

Details of a Python-based HGAA implementation are given and preliminary evaluation results are discussed. Each analysis done to date, shows a linear relationship with the number of attributes or number policy AST nodes, suggesting linear scalability. A number of possible optimizations are mentioned, including precomputing policy ASTs and intermediate symbol tables. Further evaluation of the architecture as a whole and under more diverse scenarios is needed in future work as well as comparisons to solutions utilizing generic architectures and standards (i.e. XACML, SAML, etc.) but preliminary results are promising.

Other directions for future work include extending the AC specification and architecture to support the delegation strategies discussed in [14] and introducing more flexible revocation rules. Creating an administrative model for HGABAC (or possibly utilizing the GURA model presented in [7]) would allow for the creation of administrative services specified in HGAA but currently left unimplemented.

REFERENCES

[1] Anne Anderson, Anthony Nadalin, B Parducci, D Engovatov, H Lockhart, M Kudo, P Humenn, S Godik, S Anderson, S Crocker, et al. 2003. *eXtensible Access Control Markup Language (XACML) Version 1.0.* Technical Report. OASIS.

[2] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. 2005. *Uniform Resource Identifier (URI): Generic Syntax.* STD 66. RFC Editor. http://www.rfc-editor.org/rfc/rfc3986.txt http://www.rfc-editor.org/rfc/rfc3986.txt.

[3] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. 2017. ABAC with Group Attributes and Attribute Hierarchies Utilizing the Policy Machine. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control.* ACM, 17–28.

[4] S. Farrell, R. Housley, and S. Turner. 2010. *An Internet Attribute Certificate Profile for Authorization.* RFC 5755. RFC Editor. http://www.rfc-editor.org/rfc/rfc5755.txt

[5] David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. 2011. The Policy Machine: A Novel Architecture and Framework for Access Control Policy Specification and Enforcement. *Journal of Systems Architecture* 57, 4 (2011), 412–424.

[6] David Ferraiolo, Serban Gavrila, and Wayne Jansen. 2014. *Policy Machine: Features, Architecture, and Specification.* US Department of Commerce, National Institute of Standards and Technology.

[7] Maanak Gupta and Ravi Sandhu. 2016. The $GURA_G$ Administrative Model for User and Group Attribute Assignment. In *International Conference on Network and System Security.* Springer, 318–332.

[8] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. 2013. Guide to Attribute Based Access Control (ABAC) Definition and Considerations (draft). *NIST special publication* 800, 162 (2013).

[9] John Hughes and Eve Maler. 2005. *Security Assertion Markup Language (SAML) v2.0 Technical Overview.* Technical Report. OASIS. 29–38 pages. SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08.

[10] Xin Jin, Ram Krishnan, and Ravi S Sandhu. 2012. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. *DBSec* 12 (2012), 41–55.

[11] E Rissanen, H Lockhart, and T Moses. 2009. *XACML v3.0 Administration and Delegation Profile Version 1.0.* Technical Report. Committee Draft.

[12] Carlos E Rubio-Medrano, Clinton D'Souza, and Gail-Joon Ahn. 2013. Supporting Secure Collaborations with Attribute-Based Access Control. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on.* IEEE, 525–530.

[13] Daniel Servos and Sylvia L Osborn. 2014. HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control. In *International Symposium on Foundations and Practice of Security.* Springer, 187–204.

[14] Daniel Servos and Sylvia L Osborn. 2016. Strategies for Incorporating Delegation into Attribute-Based Access Control (ABAC). In *International Symposium on Foundations and Practice of Security.* Springer, 320–328.

[15] Daniel Servos and Sylvia L Osborn. 2017. Current Research and Open Problems in Attribute-Based Access Control. *ACM Computing Surveys (CSUR)* 49, 4 (2017), 65.

[16] Jakob Simon-Gaarde. Accessed: 2017-12-15. *Ladon Webservice Framework.* http://ladonize.org

[17] SQLAlchemy. Accessed: 2017-12-15. *SQLAlchemy - The Database Toolkit for Python.* https://www.sqlalchemy.org/

[18] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. 2000. *AAA Authorization Framework.* RFC 2904. RFC Editor. http://www.rfc-editor.org/rfc/rfc2904.txt

[19] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. 2004. A Logic-Based Framework for Attribute Based Access Control. In *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering.* ACM, 45–55.

A ATTRIBUTE CERTIFICATE BYTE ENCODING