Atomic Authorization

Jacob Dilles

George Mason University

2009

Abstract

This document details one solution to the problem of atomic authorization. The focus of this paper is on a document standard called the Authorization Proof; it's anatomy and function, and a method by which this Authorization Proof may be utilized to provide a cryptographically strong, portable, scalable, securely federated authorization service.

Table of Contents

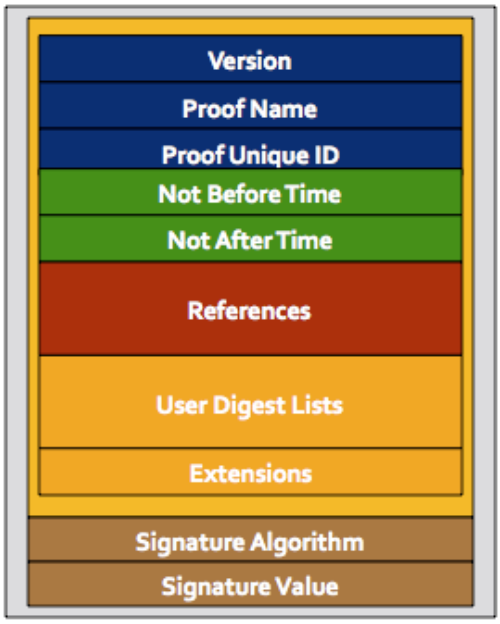
Authorization as a Service, Introduction

Certificate-based PKI credentials provide a scalable, decentralized, cryptographically secure mechanism to insure the authenticity of a computerized transaction. They are rapidly spanning many sectors: Over one million certificates issued to SSL enabled websites handle billions of secure sessions daily; The United States Department of Defense has issued over 17 million Common Access Cards (CAC) to enlisted and civilian personnel - each holding one or more X.509 certificates which control access to computer networks, enable users to sign documents electronically, and enter controlled facilities; An estimated 60 million ePassports, each with an identity certificate, were in circulation around the world in 2007, with millions more issued each year. Incorporating these credentials into existing systems significantly increases the assurance and security of n-factor authentication[1].

It is now possible for applications to implement strong authorization using identity authentication certificates that are already in place. However, beyond ownership, or possession, of the credential, a certificate contains no information on the credential holder's authority to engage in a transaction, so the challenge is making sure these applications can securely differentiate authorization rights between users. Lack of an authorization-information standard has led to proliferation of non-interoperable proprietary technologies; application-specific solutions usually involve a database lookup, for example the OPM CVS, or attributes for RBAC mechanisms like Microsoft Active Directory. Many of these implementations are significantly less secure than the credentials with which they authorize. In order to consistently and repeatability secure application authorizations in a way comparable to strong authentication, rights must be published and secured independently of the applications that rely upon them. This approach is called atomic authorization.

[1] The terms authentication and authorization are frequently confused: Since the first time-sharing multi-user computer systems were developed in the 1960's, access control authorization has been integrated into to user authentication, and the model has remained essentially unchanged. This has lead many to the assumption that the problem of authentication is identical to that of authorization. For many new applications, the problems are distinct and independent.

The Authorization Proof



The Authorization Proof one solution to the problem of atomic authorization. It is based on an ASN1 specification initially drafted in 2006 (Russell, Braceland and Lloyd). While they are not identical, Proofs share many qualities and attributes with the X.509 Certificate standard (RFC-2459)– including DER encoding[2] – and the process of implementing an encoder-decoder is analogous. A Proof contains five basic sections: identification, validity, references, credentials, and a signature. This is an overview of the key elements and their function.

Proof Identification Section

The Proof Identification Section is fairly straightforward; all components have counterparts in the X.509 certificate. This section consists of:

⇒ Version – The version number of the Proof standard. V1009a.

⇒ Proof Name – The Distinguished Name of the Proof.

⇒ Issuer Information – The distinguished name, Proof ID, and a key identifier of the Proof Authority which issued the proof

⇒ Serial number – a unique identifier which does not change between publications

The information within this section is hashed to form the Proof ID, a globally unique resource identifier.

Valid Duration Section

The Validity Section consists of three dates, which are (in chronological order):

[2] A complete specification in the ASN1 annotation style may be found in Appendix A.

⇒ Not Before - The Proof should not be accepted before this date

⇒ Next Available - The next Proof should be published on this date

⇒ Not After - The Proof expires after this date

References

The References section allows for federation across multiple groups, agencies, organizations, or priorities levels:

⇒ Peer References

⇒ Subordinate References

User Digest List

The User Digest List holds hashes of user credentials, and is used to determine authorization status.

Signature

Like a certificate, Proofs are digitally signed to give them cryptographic authority and ensure data integrity. See validation.

Proof Distinguished Name

The Proof's Distinguished Name (DN) serves to identify the resource for which the credentials within the proof are authorized to access, and it designates the directory in which the Proof should be stored. The DN is tokenized as a directory name:

Like a certificate, the DN is not authoritative, but it is a convenient way to access

```
<DistinguishedName>
        ou=Gate A Access, ou=Access, ou=Security, dc=Blue, dc=Corp
</DistinguishedName>
```

Figure 0 - Authorization Proof Distinguished Name, in XML notation

and store proofs. The XML-compliant DN in Figure 2 refers to a Proof that authorizes access to “Gate A” for the “Blue” corporation.

Proof Identifier

The Proof Identifier (PID) is the globally unique hash value that is used to specify a Proof in the context of an authorization. Selecting a hash function with sufficient key length (i.e. SHA-256) will reduce the chance of a collision, and when paired with the Proof Name in a request, the probability is statistically insignificant. A useful collision would be needed for a rouge Proof Authority to conduct a redirection attack.

```
<ProofID>
        30 7e 04 16 04 14 96 55   2a 76 37 ae ed 17 0b 07
        9a 56 3a 2d 5a 55 3a c1   f9 2a 30 61 31 23 30 21
        06 03 55 04 0b 13 1a 4d   61 73 74 65 72 20 41 75
        74 68 6f 72 69 7a 61 74   69 6f 6e 20 47 72 6f 75
        70 31 0e 30 0c 06 03 55   04 0b 13 05 4f 7a 6f 6e
        65 31 14 30 12 06 0a 09   92 26 89 93 f2 2c 64 01
        19 13 04 42 6c 75 65 31   14 30 12 06 0a 09 92 26
        89 93 f2 2c 64 01 19 13   04 43 6f 72 70 02 01 15
</ProofID>
```

Figure 0 - Sample Proof ID, in XML notation

The PID is a binary value. In persistent storage, or where it may have to read by a human, it should be encoded in hexadecimal. With whitespace and line breaks ignored, the preferred format is groups of 16 bit words, with a space between each bit, and two spaces between bytes. During RPC transactions, Base64 encoding is sufficient, so long as it is converted to HEX when the transaction is logged.

Not Before, Next Available, Not After

A certificate has an optimistically long validity period before it expires, and relies on the Certificate Revocation List (CRL) infrastructure to distribute information about certificates that should be prematurely revoked. Recognizing this method’s flaws in scalability, reliability, and speed, the Authorization Proof takes an entirely different, pro-active approach.

A publishing schedule, determined during the planning phase of Atomic Authorization implementation, dictates the time between the **Not Before** and **Next Available** dates. Every publication cycle produces a new Proof with the same Proof ID as the last, but with **Next Available** and **Not After** dates in the future. This schedule is designed in such a way that extremely sensitive privileges may be published frequently – i.e. every two minutes. Less important Proofs can have a much longer validity period, on the order of months to years. The time between **Next Available** and **Not After** is forms a grace period that allows authorizations to continue during communications malfunction or a denial of service (D.o.S.) attack.

The ratio of cycle duration to grace period in typical applications will be close to 1:1. Where the situation dictates "Fail-Open" behavior (nuclear reactor shutdown authorization) the **Not After** date may be set many years in advance.

User Digest List

A list of credentials for authorized users is at the heart of any authorization system. In an Authorization Proof, credentials are stored as digests rather than their original form for a number of reasons:

1. Anonymity – Storing entire credentials, certificates for example, may connect otherwise inert information in a sensitive way.

2. Size – Size matters for scalability, and when using credentials other than certificates

3. Speed – Searching for a hash of data in a sorted list is much faster than searching raw data.

4. Security – Using collisions to forge a digital signature on a useable document is much more difficult with fixed-length hash-value content, especially if the digests are of digitally signed documents (i.e. certificates).

5. Versatility - Any digital data may be used as credentials (Biometrics, signed plaintext, passwords, hardware keys, etc…) interchangeably without affecting the process. The algorithm particulars are specified using a conventional OID, and may be specified according to need. The standard in digesting algorithms for security, speed, and low collision probability is SHA-512.

```
<Credential type="X509Certificate">

        MIICVzCCAcCgAwIBAgIBCTANBgkqhkiG9w0BAQUFADASMRAwDgYDVQQDEwdCbHVlIENBMB4XDTA4MTExNDE4ND
        UwNloXDTEzMTExNDE4NDUwNlowVDEUMBIGCgmSJomT8ixkARkWBGNvcnAxFDASBgoJkiaJk/IsZAEZFgRibHVl
        MQ8wDQYDVQQLEwZwZW9wbGUxFTATBgNVBAMTDEJsdWUgQWRtaW4gMTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgY
        kCgYEAtJev1JSNZs7+/qmbkLGaqpiMjOv3jQWMOhhlUXXr0vjWNtiVpzUkYLHGfF5tLs1Zp0e3zQ4enO0txdSR
        ZU0ZhhTgHQRQNxS9Lqxhrh9A4EKU0MEVuapSHlLUBiT/IX1MrIv1bG7VQguGZnipH3aRsvHSsYx+lklvu80HJo
        tymQ0CAwEAAaN7MHkwOgYDVR0jBDMwMYAUoflje9N/BGpeca5bQmnrnAIIxeKhFqQUMBIxEDAOBgNVBAMTB0Js
        dWUgQ0GCAQEwHQYDVR0OBBYEFBg9t7mGcUde44Nh2SLRio/p0FGUMAwGA1UdEwEB/wQCMAAwDgYDVR0PAQH/BA
        QDAgP4MA0GCSqGSIb3DQEBBQUAA4GBACkznMnjzyRjauPYBRo1LSdamzndwaILn9u6YHz0dQdamr94IY7ly4tz
        aodbo3PYogEyC7ilU8Yr35pfdcjyhOJDRLK3kp2zGqwebl9iYbELhqpQUYSsTnb8Gajx+VMMDUJHcWRgz4J/Vc
        MtaIh7+YWzWdsuIa39OpP2Nl7Sv5MW

</Credential>
```

Figure 0 - Sample Credential Element in XML notation, encoded using Base64, before digesting.

Validation

There are a number of ways that a Proof must be validated before accepting it:

1. Signature validation. A key, signature, and algorithm are provided in the Proof. Re-signing the data in the Proof with the specified algorithm must produce the same signature, the same way signatures are verified in X.509 certificates.

2. Signature authentication. A pre-shared trust list removes dependence on existing certificate infrastructure. The key provided must be authenticated to a trusted authority.

3. ID conformation. The indicated PID must match the calculated and expected (either configured or referenced) PID

When these three conditions are met, the Authorization Proof has been successfully validated, and it's contents may be regarded as genuine. The entire process is very quick thanks to the efficiency of hashing algorithms.

Authorization Infrastructure

Like a certificate, the Authorization Proof alone is of little value. A support infrastructure of some sort is required to realize it's potential. This infrastructure consists of four primary components:

1. Administration Interface – To allow configuration of Authorization Proof publishing and authorization of credentials
2. Authorization Authority – To publish the Authorization Proof on a regular schedule, and to host these proofs for retrieval
3. Authorization-enabled application – To retrieve Authorization Proofs from the Authority
4. Local Interface – To accept credentials from a user and take action based on their status

The first and second component may be integrated when physical access to the authority is possible. This may not always be the case, such as a drop box directory server working from inside a vault. The third and fourth component may be integrated in some cases, such as a web application, however they will be discreet components for physical systems like gate access controls.

Federation

One of the Authorization Proof's greatest strengths is federation. It has the ability to include reference to other Proofs and information on where to obtain them. This allows decentralized administration by enabling trust relationships between organizations. For example, in a fictional Green and Blue corporation:

1. Green's staff needs access to Blue's systems, so both establish a policy in which Blue trusts Green to properly vet employees who request access to specific information. Both organizations implement a proof-based authorization scheme.
2. To create the trust link between the two organizations, the Blue administrators

insert a reference to the Green Proof into their Blue-Proof. The Blue administrators do not need nor want any know knowledge of the Green employees in the Green Proof; They are all managed by the Green administrators

3. When the Green user accesses the Blue application, the Blue server is queried to determine if the user is authorized.

4. The Blue server looks up the application proof on Blue's directory server and checks for a reference to the user. When the user is not found, it sees that the proof contains a reference allowing Green to issue authorizations through the use of a third party proof.

5. Knowing that Green can issue authorizations, the Blue server checks the Green directory server for a valid proof. It checks to see if the user is listed on it for an access role.

Since the Proofs of both Green and Blue may be fetched eagerly and cached, the entire series of transactions is fast and efficient. Federation works across branches in one organization, across multiple organizations, and across countries and continents. The Authorization Proof structure does not limit the number or depth of references (although the implementation should set it by policy).

General Implementation Recommendations

- Secure logging – all authorizations, publications, transfers, and Proof updates should be securely recorded in a manner that facilitates simple and accurate auditing.

- Speed – Implementation should be scale well, use a minimum of resources, and be reasonably fast.

- Resilient – Failure to find a newer Authorization Proof should not prevent the system from possessing authorizing requests

- Encapsulation – An implementation should be encapsulated to discrete and independent components and connected using open interface standards to provide

maximum security and upgradability.

- Bandwidth – The application that parses proofs must retrieve copies as seldom as prudently possible to reduce bandwidth loads, similarly the directory server should be capable of handling the transfer of Proofs.
- Security boundary – Developers should consider the security boundary surrounding each component. For instance, the Proof Authority should not be readily accessed from outside the boundary
- Ease of integration – The implementation should easily integrate with current applications, preferably with canned APIs.

Administration Interface

The administration interface used to add, select, and remove the certificates that are hashed into proofs. To avoid a "chicken and egg" scenario, some root credentials must be built into the system, however it is preferable that they should only be used to establish primary administrative credentials, after which the root credentials should be permanently disabled or escrowed. All administrative actions through this interface should require at least two-factor (ownership and knowledge) authentication.

Basic controls include:

- Add, remove, update a user certificate in the System
- Create or remove a Proof from the System
- Add or remove a user from in a Proof
- Add or remove a reference to a Proof in a Proof
- Set the publication policy in a Proof
- Export the attributes of a proof (it's DN, ID, etc…) for use in an application

However there is potential for sophisticated functions such as:

- Group management of Proofs and Users
- Acquisition and import of certificates from external sources
- Integration with existing identity providers
- Actions with option to require the approval of more than one administrator

The simplest implementation of an administration interface is one integrated into the Authorization Authority, and accessed on the physical machine. This is not a viable option where the organization's security policy makes physical access to the Authority impractical, in which case the interface would be a separate software application run outside of the Authority security boundary. The remote interface must carry secure communication with the Authority, and must be immune to DOS, Replay, and impersonation attacks.

Authorization Authority

The Authorization Authority is responsible for holding certificate credentials in a secure manner, publishing Proofs, and placing published Proofs into a publicly accessible directory. The back end implementation of the Authority is irrelevant as long as it is tamper evident and encrypted, flat files, directories, and databases can meet these criteria. It runs in a similar form and fashion to a Certificate Authority, and should take the same precautions:

- Check for data integrity frequently
- Synchronize with network time, and ensure it moves forward only
- Log all transactions, modifications, and publications in a cryptographically verifiable format
- Proof directory should be read-only, with no write access except for the authority.

Optional features may include:

- Use of cryptographic hardware for encryption and signing
- Tracking identity certificates which may be re-issued for change of rank, address, marital status, etc…
- Ability to load share between Authorities

Authorization Application

The authorization Application is the most common component in the Authorization Proof system, like the certificate validater is the most common component in the X.500 system. It may be incorporated with the application that desires authorization information, or it may be a separate component integrated through a published interface. In either case, the logic for parsing a Proof is the same.

Application Logic

An Authorization-Enabled application has two phases: maintenance and authorization.

Phase A: Maintenance. Maintain set of current authorization data:

1. Maintain a list of trusted directories, and their public keys
2. Maintain a list of Authorization Proofs and their Proof IDs
3. At the next-available time (specified within the Proof), retrieve the newly published Proof from the appropriate directory
4. Decode the new Proof ASN1 data
5. Ensure it was signed by a trusted authority
6. Verify the Proof ID
7. Traverse any peer references specified within the proof, repeating steps 4-7
8. Store the newest copies of the Proof data

Phase B: Authorization. To authorize a credential to a resource:

1. Look up the appropriate Proof in the data store
2. Hash the credential according to the specified hash algorithm
3. Search the Proof for the hash value. If found, the credential is authorized
4. If not found, recursively traverse all peers in the tree, repeating steps 2 and 3

5. If still not found, credential is not authorized

A dedicated Authorization Server would enable securely federated authorizations while isolating the developer's application from the Authorization Proof and it's semantics. The first requirement is a published interface defining the transaction between the "client" application and the Authorization "server". There are several options for this interface, such as a lightweight would be to use an XML based RPC protocol such as

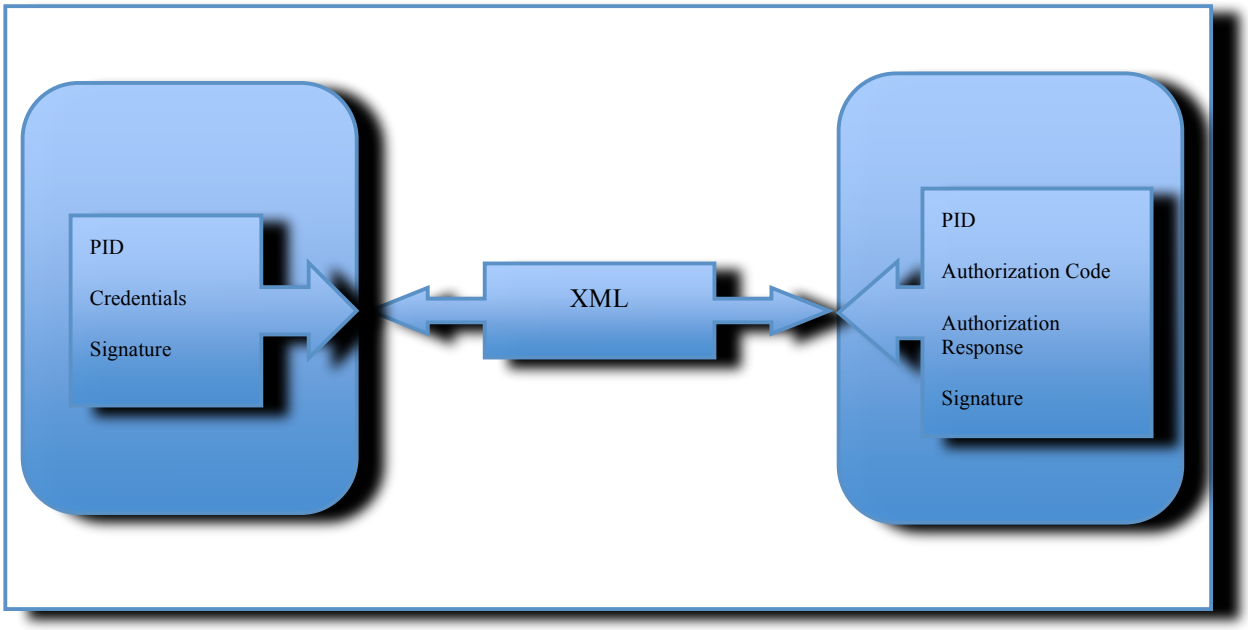


Figure 9 - The authorization sequence

SOAP, or the more complex standard SAML. In the SAML case, the Authorization Server would take the form of an Identity Provider.

Comparison of Alternatives

As noted above, the Authorization Proof is just one solution that provides Atomic Authorization. It has both advantages and disadvantages when compared with other solutions.

Advantages	Disadvantages
Attribute Certificates	
▪ ACs are digitally signed, and therefore meet the standard of atomic authorization. ▪ Can be managed separately from the entity credential resolving some card management issues.	▪ Requires central management of authorizations. ▪ Requires complicated policy framework to work in a large environment. ▪ No products readily available to either manage attribute certificates or process them. ▪ Require many certificates per user, exceeding limits of embedded devices. ▪ No contract between transportation method for ACs and the application owner.
Certificate Extensions	
▪ X.509 certificates and their extensions are signed, therefore they provide atomic authorization. ▪ Most Certification Authorities are already designed to manage policy identifiers which can be used for authorization.	▪ Every change to an entity certificate requires changes to the users credentials, creating a distribution nightmare ▪ Application owners have to divest their authorization management to identity providers. ▪ List what each user is authorized to do, can connect privileges in a sensitive way.
Authorization Proof	
▪ Proofs digitally signed, and therefore meet the standard for atomic authorization. ▪ Can be managed securely in a distributed mixed security environment. ▪ Administration and management can be delegated to application owners, across organizations, and between countries. ▪ Proofs carry no extractable information about authorized certificates, maintaining anonymity. ▪ Systematic publication eliminates possibility for repudiation. ▪ Integrates with existing technologies such as LDAP, SOAP, SAML, etc.	▪ Requires authorization policy to be defined in advance, which can be a challenge. ▪ Although integrates with mixed environment, is not directly compatible with existing applications

Conclusion

Using the Authorization Proof to publish authorization information independently of identity providers and the applications themselves, developers can implement transactional atomic authorizations that are as strong as their authentications, using identity authentication certificates that are already in place. The Authorization Proof may be applied to physical access, network applications, and computer protocols, and is the next step in securing the digital age.

JSD

Appendix A – Authorization Proof Specification

```
 --Internal Version: 1.009a

AuthorizationProofv1

   { iso(1) identified-organization(3) dod(6) internet(1) private(4)

     enterprise(1) mag(26135) prod(1) ozone(1) proofv1(1) }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

   -- EXPORTS All

   -- The types and values defined in this module are exported for use

   -- in the other ASN.1 modules.  Other applications may use them for

   -- their own purposes.

   IMPORTS


     -- Imports from PKIXImplicit88

         DistributionPointName, KeyIdentifier

            FROM PKIX1Implicit88

                   { iso(1) identified-organization(3) dod(6) internet(1)

                     security(5) mechanisms(5) pkix(7) id-mod(0)

                     id-pkix1-implicit-88(2)}

     -- Imports from RFC 3280 [PROFILE], Appendix A.1

         AlgorithmIdentifier, Certificate, Name, Extensions

            FROM PKIX1Explicit88

                   { iso(1) identified-organization(3) dod(6)

                     internet(1) security(5) mechanisms(5) pkix(7)

                     mod(0) pkix1-explicit(18) }

     -- Imports from RFC 3852 [PROFILE]

         SubjectKeyIdentifier, Digest

              FROM CryptographicMessageSyntax2004

                   { iso(1) member-body(2) us(840) rsadsi(113549)

                     pkcs(1) pkcs-9(9) smime(16) modules(0)

                     cms-2004(24) };

   ---------------------------------

   -- Authorization Proof Definition

   ---------------------------------

   AuthorizationProof  ::=  SEQUENCE  {

        tbsAuthorizationProof TBSAuthorizationProof,

        signatureAlgorithm    AlgorithmIdentifier,

        signatureValue        BIT STRING  }

   TBSAuthorizationProof  ::=  SEQUENCE  {
```

```
        version                 INTEGER,
            issuer                      ProofReference,                 -- Root Authority reference
            subject                 ProofReference,                     -- reference to self
            validityPeriod              ValidityPeriod,
        references              ReferenceMap,               -- peer and subordinate references]
        entityDigestList[0]     DigestList OPTIONAL,
        extensions [1]          Extensions OPTIONAL
    }
    ValidityPeriod ::= SEQUENCE {
        notBefore               GeneralizedTime,
            nextAvailable               GeneralizedTime,
        notAfter                GeneralizedTime
    }
    ProofIdentifier ::= SEQUENCE {
        authorityKeyIdentifier KeyIdentifier,                   -- OCTET STRING, Unsure could be DB Generated or set in a
file
        issuerDN                Name,                           -- Proof's issuerDN
        serialNumber            INTEGER }                       -- serialNumber of Proof (remains the same for each
issue)
    ReferenceMap ::= SEQUENCE {
            superior            AuthorizationReference,                         -- Parent node in authorization path
        peer [0]                SET OF AuthorizationReference OPTIONAL, -- sibling/cousin or external node
        subordinate [1]         SET OF AuthorizationReference OPTIONAL  -- Child node in authorization path
    }
    ProofDistributionPoints ::= SEQUENCE SIZE (1..MAX) OF DistributionPointName
    ProofReference      ::= SEQUENCE {
            referenceDN                 Name,                               -- X.500 naming
            proofID             ProofIdentifier,
            signedProofID               BIT STRING,
            distributionPoint   ProofDistributionPoints,     -- Follows certificate distribution point format
            certificate                 [0] Certificate OPTIONAL }   -- Added only for Assertion and Root Authority
proofs
    AuthorizationReference  ::= SEQUENCE {
            subject             ProofReference,
            issuer                      ProofReference  }
    DigestList ::=  SEQUENCE  {
        digestAlgorithm     AlgorithmIdentifier ,          --The algorithm used to digest the object in ObjectReference.
By default it is SHA256
        digests             SET OF ObjectReference}
    ObjectReference ::= SEQUENCE {
        subjectKeyIdentifier [0]        SubjectKeyIdentifier OPTIONAL,
        objectDigest                    Digest }
END
```

Appendix B – Glossary

Pending Review

Appendix C – References

Works Cited

108th Congress. "Intelligence Reform and Terrorism Prevention Act of 2004." Intelligence Reform and Terrorism Prevention Act of 2004. Washington: United States of America, 17 December 2004.

Dillaman, K. Implementation of Centralized Clearance Database. Notice. Washington, DC: U.S. Office of Personnel Management , 2006.

Housley, Russ and Tim Polk. Planning for PKI. New York: Wiley Computer Publishing, 2001.

RFC-2459. "Internet X.509 Public Key Infrastructure Certificate and CRL Profile." January 1999. Network Working Group Standards Track. Ed. R. Housley, et al. The Internet Society. January 2008 <http://www.ietf.org/rfc/rfc2459.txt>.

Russell, William C., et al. Systems, Devices and Methods for Managing Cryptographic Authorizations. Ed. LLP. Crowell & Moring. US: Patent 12101363. 11 April 2008.