TLS Working Group
INTERNET-DRAFT
Expires August 1998

Stephen Farrell
SSE
February 28, 1998

TLS extensions for AttributeCertificate based authorization

<draft-ietf-tls-attr-cert-00.txt>

Status of this memo

Abstract

Authorization support is required for various internet protocols including TLS (and its cosumers), IPSEC and others. This document presents requirements for providing such support as well as an outline specification for AttributeCertificate based authorization as an extension to the TLS protocol.

Future versions of this specification will define an encoding for the data structures required (ASN.1 or not) and will refine the description of the use AttributeCertificates in the TLS protocol.

1. Introduction

The provision of authentication, data integrity and confidentiality services for current internet protocols is well understood and many secure transports are defined (e.g. TLS, IPSEC etc). In many applications these services are not sufficient to provide the type of authorization services required.

AttributeCertificates (ACs) provide a method of overcoming these problems. An AC is a structure which is similar to an X.509 public key certificate with the main difference being that it contains no public key. The AC typically contains group membership, role, clearance and other access control information associated with the AC owner.

In conjunction with authentication services ACs provide the means to securely transport authorization information to applications.

The remainder of this document assumes that the reader is familiar with the problems with supporting authorization in a network environment.

2. Requirements

The following are the requirements which ACs are intended to meet:

R1. Support for short-lived ACs is required. Typical validity periods would be measured in hours, as opposed to months for X.509 certificates. This means that ACs can be used without mandating a revocation scheme.

R2. ACs may be valid for a set of durations. For example, from 9am to 1pm and from 2pm to 6pm, but not between 1pm and 2pm.

R3. Some standard attribute types should be defined which can be contained within ACs, for example "access identity", "group", "role" "clearance", etc.

R4. Issuers of ACs should be able to define their own attribute types for use within closed domains.

R5. It should be possible to "target" an AC. Effectively this means that a given AC may be "targetted" at one, or a number of, application servers/services in the sense that a trustworthy non-target will not use (parts of) the AC for authorization decisions.

R6. It should be possible for a server to delegate an AC when it acts as a client (for another server) on behalf of the AC owner.

R7. Delegation should be controlled, so that not every AC is delegatable and so that a given delegatable AC can only be delegated in a targetted fashion.

R8. Delegation should support chains of delegation where more than one intermediate server is used.

R9. ACs should support the encryption of some, or all, attributes (e.g. passwords for legacy applications). It should be possible for such an encrypted attribute to be deciphered by an appropriate server even where the AC has not been received directly from the AC owner (i.e. where the AC is delegated).

R10. ACs should be defined so that they can either be "pushed" by the client to the server, or "pulled" by the server from a network service (whether the AC issuer or a directory service).

R11. Attribute types should be defined so that it is possible for an AC verifier to distinguish between e.g. the "Administrators group" as defined by SSE and the "Administrators group" as defined by Widgets inc.

R12. ACs should support anonymity in the sense that certain ACs should be usable even when they don=92t contain a name for the AC owner. (Note however, that some authentication services may still expose this information to the AC verifier.)

R13. ACs should support audit mechanisms (e.g. through the use of audit identities).

R14. ACs should support billing mechanisms (e.g. through the use of charging identities).

3. Operational Models

Some internet protocols and environments may work best when a client "pushes" an AC to a server. In other cases it is more suitable for the client simply to authenticate

to the server and for the server to request the client=92s AC from another network service.

Each of the above is suitable in different circumstances. For example, the "pull" model can often be implemented without changes to the client, whilst the "push" model requires no new connections to be established (which may mean that firewalls don=92t have to be reconfigured).

Supporting these requirements and operational models requires that we define some AC controls in addition to the AC contents.

4. AC contents

The following describes contents of an AC.

<<Current version is informally defined. Issues to do with encoding (e.g. ASN.1 or not) are for later>>

4.1 Overview

An AC consists of the following fields:

=

```
attributecert  ::=3D  ISS issuer
                      SER serial
                      CRE creationtime
                      VAL validity
                      AID auditid
                      [OWN owner ]
                      ATT attributes
                      [ TRG targetting ]
                      ALG algorithm
                      SIG signature
```

=

Where the issuer is the name of the entity who produced (signed) the AC, the serial is a number so that the issuer & serial pair form a unique identifier for the AC. Note that there is no restriction that the issuer and owner be distinct - this specification allows entitites

to simply assert that they possess certain attributes (the attribute values themselves could actually contain some proof about the association!).

<<Information about the issuer public key certificate which should be used to verify the signature could be added to reduce searching at the server.>>

The validity field contains a set of (possibly overlapping) durations during which the AC is potentially valid. The auditid is the identity, for audit purposes, of the entity with whom the attributes are associated. The (optional) owner field names the entity with whom the attributes are associated. The attributes field contains a set of attribute values associated with the AC owner. The optional targetting field allows for cases where the AC is not to be valid everywhere or for support of controlled delegation. The algorithm and signature fields contain the signing algorithm and signature of the AC issuer over the AC contents.
=

```
issuer ::=3D STRING
serial ::=3D  NUMBER
creationtime ::=3D TIME
validity ::=3D period  [ validity  ]
period ::=3D notbefore notafter
notafter ::=3D TIME
notbefore ::=3D TIME
```

```
auditid ::=3D  STRING
owner ::=3D STRING [ NULL | owner ]
algorithm ::=3D STRING
signature ::=3D BITS
=
```

4.2 Attributes

The AC contains an unordered set of attributes (i.e. more than one instance can occur). Each attribute has a type (e.g. group, role, clearance); an optional policy authority (defaulting to the AC issuer if not present); a criticality and an (unordered) set of values.

The policy authority names the entity who defines the semantics for this attribute type and value. In this way RESTRICTED clearance as defined by Siemens can be distinguished from the RESTRICTED clearance as defined by NATO.

The criticality flag (default FALSE) if set, indicates that this AC must be ignored for authorization purposes unless the processing entity can handle the semantics of the attribute. The criticality flag allows for the implementation of restrictions.

```
attributes ::=3D attribute [ attributes ]
attribute ::=3D AT type [ policyauthority ] =

                [ CT criticality ] atvalues
type ::=3D STRING
policyauthority ::=3D STRING
criticality ::=3D BOOLEAN
```

```
atvalues ::=3D atvalue [ atvalues ]
atvalue ::=3D STRING
=
```

4.2.1 Basic Attribute Types

Some basic attribute types are defined in this section. The intention here is simply to promote interoperability (so servers know where to find the values in the AC) and not to define rigourous semantics for these attribute types.

Unless otherwise stated, all the basic attribute types may be multi-valued. In all such cases, it is the responsibility of the server to select the appropriate value(s) when making access decisions.

Unless otherwise stated, all the basic attribute types may be critical at the discretion of the AC issuer.

<<there's no case so far where the type must be critical to be useful>>

This specification only defines attribute types, there is (currently) no intention to define standard attribute values.

<<Though some such values would be useful. Note also that their syntax is TBS as it depends on the ASN.1 or not issue.>>

4.2.1.1 audit identity

An audit identity is the identity of the AC owner for audit purposes. This may be in the form of a name (e.g. "Stephen Farrell from SSE") or a number (e.g. "12357886 from CN=3DAC Issuer;O=3Dsse;C=3DIE"). Only a single value of audit identity is allowed.

Note that the latter form allows audit records to be written so that the cooperation of the AC issuer is required in order to identify the individual involved. (This may be required by data protection and labour law in some countries). Note that there is no requirement

that such numbers are permanently associated with the individual.

4.2.1.2 access identity

An access identity names the AC owner for authorization purposes. Examples are "stephen.farrell@sse.ie", "SFARRELL", "user123".

4.2.1.3 charging identity

This attribute allows an AC to contain information =

intended to be used for billing purposes (though this specification says nothing else about billing systems). Examples are "Stephen Farrell", "Security Business Unit".

4.2.1.4 group

This attribute contains group membership information. The groups may be OS or application defined groups.

<<might be more useful to define OS specific group atttributes, e.g. NTDOMAINGROUP "Administrators in SSE.IE" or unix groups "100">>

4.2.1.5 role

This attribute contains information about the role(s) in which the AC owner may be acting. Examples are "Duty Officer", "Senior Accountant".

4.2.1.6 clearance

The clearance attribute contains information about the clearance(s) of the AC owner. Examples are "SECRET", "COMPANY CONFIDENTIAL".

4.2.1.7 encrypted attributes

Some environments call for the use of encrypted attributes. This is commonly required in order to centralise the administration of passwords for legacy

applications which are accessed via the underlying secure transport.

As attributes are encryted the issue of transporting/agreeing the attribute encryption key arises. Many different methods may be used for this, however, this specification defines one method, which allows the AC issuer to agree an attribute encryption key with a specific target. (This method is suitable for handling legacy passwords.)

The method is as follows:

For each (group of) attributes to be encrypted for a particular (set of) target(s):

1 Encode the attributes to be encrypted (including type, policy authority, criticality and value(s))
2 Generate a random attribute encryption key (AEK, triple-DES default)
3 Encrypt encoded attributes
4 For each target in the current set:
  4.1 Using issuer's and target's D-H keys agree a key encrypting key (KEK,triple-DES default)
  4.2 Use this KEK to encrypt the attribute encryption key
5 The target name(s), D-H value, encrypted AEK and encrypted attribute(s) form a single value of the attribute EncryptedAttributes

The encoding is illustrated below:

```
Attrtibute Type::=3D EncryptedAttributes
Policy Authority::=3D <<AC issuer>>
Criticality::=3D <<according to policy but typically
                 FALSE>>
Value0::=3D TG <<target>> DH <<D-H>> <<encrypted AEK>>
           <<ciphertext >>
Value1::=3D ...
```

<<it may be best if pkcs#7/CMS structures are actually used for the values - same problem so why not re-use the solution? (basically, because it determines the encoding issue which should be dealt with later)>>

4.3 Targetting

If an AC contains an owner field and no targetting field then the AC must only be used for authorization purposes when the AC is received directly from the owner via an

authenticated transport mechanism. The server must therefore compare the client identity as provided by the (peer-entity) authentication service with the AC owner field.

For cases where more fine grained control of authorization is required (e.g. an AC which should only be used by printing applications) or where delegation is to be supported (without granting all privileges to the delegate) then the targetting field allows the AC issuer to control the "coverage" of the AC (i.e. limit the places where the AC should be used for authorization).

The targetting field can specify a set of "direct" targets. Only those services specified are intended to be able to use the AC for authorization purposes. Such direct targets are not allowed to delegate the AC.

Sets of delegate services may also be named. Such services may use the AC for authorization purposes and are also allowed (by the issuer) to act as delegates (i.e. to re-use the AC) whenever they act as a client for another named service.

In addition to naming specific servers (e.g. "http://www.sse.ie"), groups of servers may be named and used to control targetting (e.g. "SSE web servers"). The interpretation of such server group names is a local matter for the AC verifier. Such names are termed service group names.

```
targetting ::=3D DR services  [ delegates ]
delegates ::=3D  DG services [ delegates ]
services ::=3D service [ services ]
service ::=3D ALL | SG servicegroupname | SN
servicename
servicegroupname ::=3D STRING
servicename ::=3D STRING
```

<<handling of wildcards will probably need to be defined, or at least explicitly allowed>>

4.4 Delegation

When an intermediate server delegates an AC to a target server then the target must be able to verify that the intermediate hasn't "stolen" the AC. This is achieved by having the each initiator and

intermediate server in the chain of delegation sign # the following data structure:

```
[ initiator (I), target (T), AC-issuer,
     AC-serial, time, nonce ]
```

<<the time and nonce may not be needed, depending on the TLS encapsulation>>

In terms of the above syntax a delegation proof is a "pairing proof" (which prooves which initiator-target pair are supposed to be involved in the connection):

```
pairingproof ::=3D PPF initiator target
                    issuer serial timestamp nonce
                    algorithm signature
initiator ::=3D STRING
target ::=3D STRING
timestamp ::=3D TIME
nonce ::=3D BITS
```

<<Question - is this a new signature or can it be part of the existing TLS handshake? Answer: existing if no need for traced delegation, otherwise a new signature is needed to provide the trace. The answer also depends on the encoding of ACs. For the present we assume a new signature is generated.>>

<<note: a version of this which doesn=92t require the client to sign would be better (assuming all servers have signature keys is reasonable). We will still need to provide a trace of delegation starting with the client somehow.>>

If traced delegation is required then a sequence of pairing proofs must be sent (with the current one left most):

```
pairingproofs:: =3D pairingproof [pairingproofs]
```

<<additional checks to be defined, e.g. initiator from current pair must be target from next pair. times shouldn=92t be out of whack by too much, etc.>>

So long as TS can verify this data then it can check (via whatever targetting is included in the AC) that IS hasn=92t

stolen the AC.

5. Sample ACs

The following (illustrative) samples give an idea of use of the above syntax.

5.1 Simple Case

Client has two attributes (two-valued role and employeeStatus). AC is =93anonymous=94, delegatable anywhere and not targetted.

```
issuer:        "CN=3DAC Iss;O=3Dsse;C=3DIE"
serial:        1234
validity:      19971220090000Z to 19971220180000Z
auditid:       1293843944 from "CN=3DAC Iss;O=3Dsse;C=3DIE"
attributes:    role: developer techsupport
               employeeStatus: permanent
algorithm:     dsaWithSHA1
signature:     "89DC...0001"H
```

5.2 Delegation Case

Owner named with same two attributes. AC is targetted and delegatable (to a group of local web server applications and two FTP servers).

```
issuer:        "CN=3DAcIss;O=3DSSE;C=3DIE"
serial:        1234
```

```
validity:      19971220090000Z to 19971220180000Z
auditid:       1293843944
owner:         fred@sse.ie
attributes:    role: developer techsupport
               employeeStatus: permanent
targetting:    DR ALL
               DG =93SSE web servers=94
               DG ftp1.sse.ie ftp2.sse.ie
algorithm:     dsaWithSHA1
signature:     "89DC...0001"H
```


6. Use of ACs in protocols

ACs occur at various steps in various protocols. In order to support a number of different transport protocols an AC carrier structure is defined which allows for requests for ACs, responses to same and for pushing ACs plus external controls (e.g. delegationProof) in a standard manner.

acnfo is the data structure which is used to "push" or "pull" an AC (or error message) plus associated control information. A set of public key certificates may also be "pushed" with an acinfo in order to assist the recipient in certificate handling.

```
acinfo ::=3D  [ attributecert ] [ pairingproofs ]
               [ pkcerts ]
pkcerts ::=3D OCTETS
```

There are a number of possible exchanges which can occur and three entities involved (client, server and AC issuer). In addition the use of a directory service as a repository for AC retrieval may be supported. Of these exchanges, the most important to embed in the TLS protocol is that between client and server. AC acquisition (from an issuer or directory) is handled as a higher layer protocol (a payload protocol from the TLS perspective).

The diagram below shows the exchanges defined which are further described in succeeding sections.

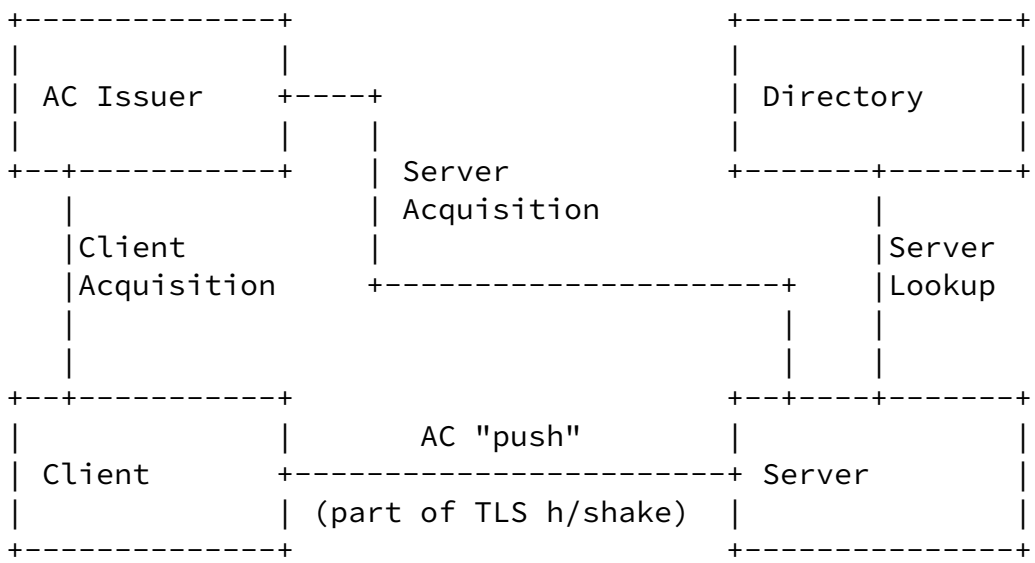

Figure 1: AC Exchanges

6.1 Client Acquisition

This exchange occurs when a client requires a new AC from an issuer. The AC issuer MUST ensure that the client is the correct one, most simply via TLS with client authentication. This exchange may occur at network login time or may happen automatically during (or before) the client handshake with a TLS server (i.e. upon receipt of the ACRequest message from the server - see section 8 below).

This exchange requires that the client be authenticated and also that the server returns an AC to the client. For these reasons (and to allow for other AC acquisition methods), this exchange cannot be embedded in the TLS handshake and so is defined as a payload protocol which uses TLS for its underlying security.

<<note: in the following TLS style message definitions are given, these are also subject to syntax choices>>

The client sends the following message to the server:

```
struct {
     opaque acinfo<1..2^24-1>
} ClientACRequest
```

ClientACRequest.acinfo may contain a template for the AC which the client wishes. The proofpairing component of the acinfo may contain the name of the server.

<<above needs further study - perhaps specific fields from the acinfo should be explicitly part of this (and other) messages>>

The server responds with:

```
struct {
     enum {
          success(0),
          success_with_changes(1),
          failure(2),
          denied(3),
          (255)
     } acstatus;
     opaque acinfo<1..2^24-1>
} ClientACResponse
```

The ClientACResponse.acinfo is only valid if ClientACResponse.acstatus is success or success_with_changes.

6.2 Server Acquisition

Server acquisition occurs where a client has established a TLS connection to the server, but hasn=92t provided (or can=92t provide) an AC. The case where the client provides a pairingproof is also supported.

<<maybe there's no point in supporting the pairingproof only case as it=92ll only happen if the client is "AC aware" in some sense - in which case why didn=92t the client push the AC? Justification for this case is that the client may not know which AC is needed, whereas the server does, but the Issuer requires that the server proove that the client has made a fresh request for this server.>>

The server sends the following message to the issuer:

```
struct {
     opaque acinfo<1..s^24-1>
} ServerACRequest
```

The attriburecert part should contain a template for an AC which the server would like to get. The pairingproof (if present) should be signed by the client.

The issuer responds with:

```
     struct {
          enum {
               success(0),
               success_with_changes(1),
               failure(2),
               denied(3),
               (255)
          } acstatus;
          opaque acinfo<1..2^24-1>
     } ServerACResponse
```

The fields here are as described for the ClientACResponse.

6.3 Server Lookup

<<TBS: just need to specify a standard directory attribute and possibly some matching rules - maybe better in another draft>>

6.4 AC "Push"

This exchange is where the server requests that the client present its AC so that an authorization decision can be made. This exchange is suitable for inclusion within the TLS handshake since client and server authentication are not always required and, if required, can be validated after the TLS handshake is complete without loss of security. This exchange is specified in section 8.

7. AC Validation (outline)

It is assumed that the acinfo structure is received with data integrity and peer entity authentication.

Validation of an AC requires that the checks described below be carried out. The output of the validation algorithm is a status (good/bad) and an optional set of attributes.

1. Signature Validation. Validation of the AC signature and related X.509 certificates.
2. Timeliness. Validation that the current time falls within one of the time periods of the AC.
3. Issuer Trust. Verify that the issuer is trusted as an AC issuer
4. Attribute Checking. For each type and policy authority verify that the issuer is trusted to determine values. Attributes which are untrusted do not cause a failure of the overall validation but MUST be ignored for access decision purposes. If the attribute is encrypted but the server does not have the decryption key then the attribute MUST NOT be used for access decisions. Critical attributes whose value cannot be interpreted MUST cause failure of the overall validation.
5. Target/Owner checking. If no owner field is present and no targetting field is present then the AC is globally delegatable.
   If an owner field is present and no targetting field is present then a server MUST reject the AC if the sender is not the owner.

When targetting is present we are essentially checking that the entity from which the AC was received was allowed to send it and that the AC is allowed to be used by this target.
The targetting information consists of the direct information and an optional set of delegate information.

If the direct check or any of the delegate checks passes then the targetting check as a whole is successful.
The direct check passes if the identity as established by the underlying authentication service =93matches=94 the owner (or no owner is specified) and the services part is either "ALL" or the current server is named in the services part or the current server is a member of one of the servicegroupnames. (How membership is determined is out of scope here.)
A delegate check succeeds if the server is named in the services part (of one of the sets) or if the server is a member of one of the servicegroupnames and the identity as established by the pairingproof is either the owner or another server from the same set of services.

8. TLS Encapsulation

<<TLS references are to draft-ietf-tls-protocol-05.txt, this section is currently quite sketchy>>

The basic requirement is to be able to include an acinfo structure into TLS handshakes.

8.1 Session Management

The TLS session state (section 7, p22) now requires a new item:

authorization information
An acinfo structure containing the authorization information for the session. This

element of the state may be null.

<<may need more state info, e.g. "AC requested", "Client indicates AC to be read from directory", etc.>>

8.2 Use of TLS Alerts

<<This section is TBS, but no new TLS alerts appear to be required. However the handling of some existing alerts may need to be extended, e.g. access_denied (49) could become non-fatal if a client has another AC which it could present.>>

8.3 Handshake Protocol Extensions

<<This section is currently a little different from section 6 - we don=92t really allow the client to "push" the AC without the server asking. Additional text is also needed for C<->I and S<->I connections >>

The basic approach is to handle the acinfo structure in the same way as X.509 certificates are handled. This means that we define an ACRequest which the server can send to the client and an ACInfo with which the client can respond. These are analagous to the existing CertificateRequest and CertificateResponse messages. The extension to the interworking diagram from TLS (section 7.3) is:

```
   Client                                                Server
   =3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D     =
    =3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=3D=
=3D=3D=3D=3D
   ClientHello            ----->                          =

                                                   ServerHello
                                                  Certificate*
                                            ServerKeyExchange*
```

```
                                          CertificateRequest*
                                                   ACRequest*
                          <-----              ServerHelloDone
   Certificate*                                                =

   ACInfo*
   ClientKeyExchange
   CertificateVerify*
   [ChangeCipherSpec]
   Finished               ----->                              =

                                        [ChangeCipherSpec]
                                                  Finished
   ApplicationData        <---->           ApplicationData
```

<<note: the new messages could be avoided if the Certificate and CertificateRequest messages were extended, however, the syntax and handling of the ACRequest and ACInfo seem sufficiently different to warrant new messages.>>

The following sections define the new messages in more detail.

8.3.1 ACRequest message

When this message will be sent:
A server can optionally request an AC from the client. This message, if sent, will immediately follow the CertificateRequest (if it is sent).

Structure of this message:

```
struct {
     opaque acinfo<1..2^24-1>
} ACRequest
```

<<note: the "opaque" above may change depending on the AC encoding decision - to an ASN.1blah or a set of TLS style fields.>>

=

8.3.2 ACInfo message

When this message will be sent:

This messag must be sent if the client has received an ACRequest message from the server. If sent, it will immediately follow the Certificate message sent by the client (if sent).

Structure of this message:

```
struct {
     opaque acinfo<1..2^24-1>
} ACInfo
```

8.4 AC Validation within TLS

<<TBS - specify where in the handshake the AC validation can occur - looks like it can happen immediately, though the actual attributes can=92t be trusted until the client authentication (if done) is complete, which would mean after the client=92s finished message>>

8.5 Preconditions for use of ACs wihtin TLS

<<TBS - specify any restrictions on ciphersuites, naming and other relevant TLS options>>

9. Conformance

Servers MUST support both "push" and "pull". Clients MUST support "push".

A signing algorithm is required. DSA-with-SHA-1 MUST be supported. For attribute encryption triple-DES-CBC-168 MUST be supported for the KEK and attribute encryption key. D-H must be suported for ageement of the KEK. <<oids TBS, should be same as TLS & S/MIME "MUST"s>>

Targetting MAY be supported. Servers which receive ACs containg targetting information but which do not support targetting MUST reject the AC unless a "direct check" succeeds. Client and Servers MUST however, be able to encode ACs containing targetting.

Encrypted attributes MAY be supported. Servers which do not support encrypted attributes should ignore their presence for access control purposes.

10. Security Considerations

For ACs to be used securely they should be transported with peer entity authentication and data integrity. The only exception is an AC which contains no owner nor targetting fields and which is therefore globally delegatable (and therefore easily stolen!).

A server which is presented with an AC which is delegatable will be able to masquerade as the AC owner to any of the other servers to which the AC can be delegated. This means that the client is effectively trusting the server to the extent that the AC is delegatable. For this reason it is recommended to limit the delegation scope of ACs to the minimum required.

A server which is presented with an encrypted attribute and the relevant decryption key will gain access to, and can possibly misuse, the plaintext attribute value(s). As the encryption key used is fixed for the duration of the AC lifetime, multiple servers may get access to the plaintext.

Author's Address

Stephen Farrell,
SSE Ltd.
FItzwilliam Court,
Leeson Close,
Dublin 2,
IRELAND

tel: +353-1-676-9089
email: stephen.farrell@sse.ie

Appendix A: Possible Extensions

AC Translation: In order to provide for a scalable solution accross domains whilst at the same time supporting private attribute types an AC translation service could be specified.

Delegation chains: Chains of delegation which involve server ACs are in priciple possible but are left for further study. Methods other than the pairingproof defined above can be defined for delegation protection (e.g. ECMA CV/PV method), these are for further study.

Query API: An API for querying the secure transport would be useful in order to allow for appilcation layer access decisions.

Delegation API: Some API will be requried in order to support delegation.