OSF DCE SIG
Request For Comments: 19.0

S. B. Fairthorne (ICL/Sesame)

December 1992

SECURITY ENHANCEMENTS FOR DCE 1.1

1. INTRODUCTION

This is the first of a potential series of DCE-RFCs proposing enhancements to the security features of DCE, based on work done by the Sesame consortium.[1] The DCE 1.1 enhancements proposed in the present DCE-RFC provide a much smaller increment than those in the so-called ''Sesame 1.X'' proposal presented at the DCE 1.1 review meeting in July 1992 — with fewer functional extensions to DCE 1.0 based on existing mechanisms and protocols. This proposal also complements other DCE-RFCs on expected security enhancements to DCE for DCE 1.1. Thus, the current proposal represents a thoughtfully integrated solution for inclusion of major Sesame features into DCE.

At the same time, a vision for the path forward from DCE 1.1 has been developed, which shows how the initial increment being offered here lays the ground work for further enhancements. These further enhancements will be the subject of future DCE-RFCs.

1.1. Objectives

The main objectives of the current proposal are:

(a) To lay the ground work for the optional use of *public key* technology in key distribution by introducing a greater *separation* between key distribution and privilege establishment mechanisms.

(b) To provide better inter-cell working using *signed PACs* (for those systems where public key technology is available).

(c) To support a greater variety of *access control policies* by enhancing the internal syntax of the PAC to be able to contain a wider variety of attributes, including customer (or OSF vendor) defined ones, and enabling the use of these in targets. This provides the first steps towards secure interworking with non-DCE systems as attributes can have semantics appropriate to such systems.

(d) To support *delegation* of a client's rights to another application for use by proxy in a way which is independent of the key technology option in use.

(e) To introduce individual *audit identities* which are separate from those used for access control for better accountability and to audit security relevant events using them.

(f) To *minimise the need to encrypt* information in the provision of secure access controls by allowing access privileges to be integrity protected, rather than confidentiality protected. This makes the solution more acceptable to European authorities.

(g) To support a *choice of encryption algorithms and key characteristics* for the protection of user data, in particular, allowing strong integrity and weak confidentiality. This makes the solution more acceptable to European authorities.

(h) To make the extended security facilities available to both *RPC and non-RPC* programs.

(i) To take the first step towards structuring the security infrastructure of DCE 1.1 target systems in a manner which improves *assurability*. This is achieved by logically separating off from the rest of the target system the functions validating incoming PACs and controlling the establishment of conversation keys.

---

1. ''Sesame'' is an acronym (with appropriate security connotations) for ''Secure European System for Applications in a Multivendor Environment''. The member companies of the Sesame consortium are Bull, ICL and SNI.

1.2. General Description of Facilities

The proposal in this DCE-RFC is a natural extension to DCE 1.0, using existing security services and protocols where possible. It makes no changes to the *Authentication Service (AS)* and *Key Distribution/Ticket Granting Service (TGS)* components of DCE 1.0 (sometimes referred to by the Kerberos terminology of *Key Distribution Center (KDC)*). Full support for existing DCE 1.0 functionality and protocols is retained (though new and enhanced security operations are provided, with new and extended arguments and returns), and the client and security server codes paths executed for those targets accepting DCE 1.0 service tickets remain virtually unchanged.

The proposal introduces optional use of public key technology — PACs can be *signed* (integrity protected using public key technology) or *sealed* (integrity protected using symmetric key technology). It also prepares for wider use of public key technology in future. Therefore, central to the proposal is a restructuring of the ticket for DCE 1.1 application services to separate the key handling and access privileges. This allows use of different key distribution options and reduces the amount of encryption used. Also, separate keys for protecting the integrity and confidentiality of user data can be used so integrity protection can be strong while confidentiality protection can be weaker. These central changes are described in Section 2.

A client's rights may be delegated to a server which may then allow these rights to be delegated further. It is possible to control whether delegation is allowed, and if so, which individual targets, or groups of targets can act as delegates for a client. The chain of delegates between the initiator and final target can also be traced. The same delegation facilities are offered independently of the key technology option used. The secret key case is essentially that described in [RFC 3.0] though there are minor changes for consistency with the public key case. (Note: DCE-RFC 3 is undergoing changes to accomodate this.) Delegation functionality and options are described in Section 3.

The proposal includes support for a wider range of privilege attributes such as those needed to support non-UNIX systems. [RFC 6.0] allows the registry to contain an extended set of attributes. This proposal allows these to be included in PACs and then made available to applications via an API. This is described in Sections 4 and 6.

The extended security services, such as the Privilege Service, audit significant security related events, though it should be noted that no changes are proposed to the AS in this DCE-RFC. Use will be made of any audit service that might be provided in response to [RFC 1.0], though if this is not available, audit records will be stored in files locally.

Some changes are also proposed to simplify administration; privileges and controls which apply to a user session can be selected automatically based on the role the user is taking in the session. Audit and management changes are described in Section 5.

The security facilities proposed are available to non-RPC as well as RPC applications. The API supported for these non-RPC applications is an extension to the GSS-API in [RFC 5.0]. The APIs affected by these enhancements are described in Section 6 below.

The extensions to the Privilege Server protocol are described in Section 7 and the updated data structures, including the PAC syntax, are described in Section 8.

1.3. Outline Architecture

The diagram below shows the main exchanges with the security services from a user logging in to accessing a server. For simplicity it does not show getting new privileges during a user session or intercell exchanges.

```
+--------+ +------+     authenticate          +------+
|enhanced|-|------|----------------------------|      |
|  DCE   | |      |     (returns TGT)         |  AS  |
| login  | |      |                           |      |
+--------+ |      |     get PS ticket         +------+
           |      |----------------------------|      |
           |client|                       +----|      |
           |      |     get privileges    |    | TGS  |
           |      |-----------------------| PS |      |
           |      | (returns PTGT & EPAC) |    |      |
           |      |                       +----|      |
           |      |    get service ticket      |      |
           |      |----------------------------|      |
           |      |                            +------+
           |      |                             |get ticket etc.
           |      |                             |for delegation
           |      |                             |
           |      |               +-----+ +------+ +-----+ +------+
           |      |               | PVF |-|      | | PVF |-|      |
           |      |  call server  +-----+ |server| +-----+ |server|
           |      |-----------------------|  A   |---------|  B   |
           |      | (sends ticket & EPAC) |      |         |      |
           |      |                       +------+         +------+
           |      |                           |                |
           |      |                       +------+         +------+
           |      |                       | DACL |         | DACL |
           +------+                       +------+         +------+
```

Initial exchanges with the existing AS and TGS components of the KDC are unchanged. An enhanced DCE login provides a command line interface to the user for authentication and selection of attributes.

The Privilege Server is extended as follows:

(a) To separate key management from privilege handling to make it easier to use different key technology options and also to reduce the amount of data which is encrypted.

The *Privilege Ticket Granting Ticket (PTGT)* and ticket formats are unchanged, so the TGS is still used to obtain service tickets, but instead of containing privilege attributes, these contain a link to an *Extended Privilege Attribute Certificate (EPAC)* (described in Section 2) which contains the attributes. It is also still possible to generate DCE 1.0 style tickets.

(b) To protect the PACs using either symmetric or asymmetric cryptography.

(c) To support a wider range of attributes — both an extended set of privileges and also controls for delegation (as described in Sections 4 and 3).

A client, when calling a server, passes the EPAC as well as the ticket. A *PAC Validation Facility (PVF)* checks the PAC and ticket at the server and only releases the keys for dialogue protection if the PAC is accepted. This is a separate component for increased assurability.

The server may use privilege attributes (in UUID form) from the PAC for access control to its own objects via DCE ACLs in the usual way. It may also obtain privilege attributes from PACs and use them in other types of authorisation mechanisms.

The first server called may go on to delegate the access privileges obtained to a further server, provided delegation controls allow this. Server B may optionally base access control decisions on the rights of server A as well as those of the original user.

2. EPACS AND KEY DISTRIBUTION OPTIONS

A major part of this proposal is a restructuring of the ticket for DCE 1.1 application servers. It is split into a ticket which handles communications security, linked to an EPAC which carries access privileges. This helps achieve the two objectives of enabling the use of confidentiality to be reduced (as the EPAC does not now need to be encrypted) and separating key distribution from access control management — the ticket design has been insulated from future design choices for particular contents of the EPAC (and vice versa). In this proposal, however, key management remains unchanged from DCE 1.0.

2.1. Tickets and EPACs

The syntax of tickets, including PTGTs, is unchanged and they are encrypted as in DCE 1.0. The authorisation data can take one of three forms depending on whether the target servers to be accessed are:

(a) DCE 1.0 servers where the tickets are as in DCE 1.0 and include privilege attributes.

(b) DCE 1.1 servers where tickets are linked to symmetrically integrity protected EPACs.

(c) DCE 1.1 servers where the linked EPAC is asymmetrically integrity protected.

For DCE 1.1 servers, the EPAC contains the user's access control privileges, along with administrative and control information. Existing identity and group privileges are all supported within an EPAC syntax which also provides for additional attribute types and controls. Existing *DCE ACLs (DACLs)* can use privileges from EPACs with no changes to DACL contents being required, subject to the choice of delegation controls. The delegation controls in the EPAC are described in Section 3.

In the symmetric case, the authorisation field in the ticket contains a hash of the EPAC to act as the link between the ticket and EPAC. The EPAC itself includes this same hash encrypted under the secret key of the *Privilege Service (PS)*, shared only with the KDC. In other words, it is sealed by the PS. This enables the PS to validate the EPAC independently of any ticket to which it is linked.

```
   PTGT or other ticket                        EPAC
+-------------------------+    +-------------------------------------+
|      | hash of |        |    | delegation and | privilege  | hash/ |
|      |  EPAC   |        |    | other controls | attributes | seal  |
+-------------------------+    +-------------------------------------+
           |                                                    |
           +----------------------------------------------------+
```

In the asymmetric case also, the EPAC includes a hash which this time is asymmetrically encrypted by the private key of the PS instead of being symmetrically encrypted in the PTGT. In other words the EPAC is now signed by the PS. The link between the PTGT and the EPAC is looser. They are linked by means of a *Ticket Requester Identifier (TRID)* held in both PTGT and EPAC. The TRID at DCE 1.1 is a random, unique identifier invented by the PS for this client (though in future releases, it may be a managed value which can identify the environment or location in which the client is operating). More than one EPAC or ticket for that requester can have the same TRID value.

```
 PTGT or other ticket                          EPAC
+----------------------+   +-----------------------------------------+
|      | TRID |        |   |delegation and | privilege | TRID |hash/|
|      |      |        |   |other controls | attributes|      |sign.|
+----------------------+   +-----------------------------------------+
           |                                              |
           +----------------------------------------------+
```

EPACs give the client's rights, tickets handle the communications security. One EPAC may be used with several tickets where the client's rights are to be passed to a number of different targets. Also, a ticket for a target could be used at different times with different EPACs belonging to the same client.

2.2. Walkthrough

The changes are further explained below in the form of a walk-through. Only access to DCE 1.1 servers is described in the following walkthrough, as access to DCE 1.0 servers is unchanged. Delegation is also not described here; it is described in Section 3.

(a) A user (principal) authenticates using DCE 1.0 code and then obtains a ticket from the TGS to talk to the PS in the usual DCE 1.0 manner.

(b) This ticket is presented to the PS to authenticate a request for a PTGT in the normal DCE 1.0 manner. It is at this point that the new DCE 1.1 functionality first comes into play. One or more of the three forms of PTGT described above can be requested for users expecting to access different service types. Enhanced client security software selects between the different options, according to negotiation, managed information held about the target servers it wishes to access or security policy.

For DCE 1.1 servers, the PTGT returned is accompanied by a separate EPAC, and possibly other control information depending on the delegation options and EPAC protection methods in use (see Section 3).

(c) When the user invokes an application, he gets a ticket (if one is not already available) from the TGS in the normal DCE 1.0 manner, using existing DCE 1.0 protocols. The TGS is blind to the different types of PTGT, inserting the PTGT's authorisation data unexamined into the ticket as at present. The EPAC and any other control information obtained by the client are not passed to the TGS.

(d) When the client and server are in the same cell, the client now engages in DCE 1.0 exchanges with the target server up to the point when the service ticket is to be exchanged. At this point, the EPAC and any associated delegation control information are also sent to the target along with the service ticket. Protocol protection and key exchange are done in the same way as in DCE 1.0 using DCE 1.0 code.

If the client calls on a server in another cell, the exchanges with the TGS in both client and foreign cells are as in DCE 1.0 (possibly updated to take into account hierarchic cells as described in [RFC 7.0]). In the case of symmetric PAC protection, the calls on the foreign PS also pass the EPAC so this can be resealed. However, when asymmetric PAC protection is being used, the exchange with the foreign PS is not needed as the PVF at the target system can use the public key of the client's PS to validate the PAC.

(e) In DCE 1.1 target servers, ticket/EPAC validation code has been separated out into an independent component: the PAC Validation Facility (PVF). The PVF is the only component at the server that shares a long term key with the KDC. This is the key used by the TGS part of the KDC to encrypt the ticket, and consequently it is the PVF that validates the ticket and its associated EPAC. Only if validation is successful will a key be released to the server's RPC runtime to enable a conversation with the client to proceed.

(f) Up to two dialogue keys can be derived from the key shared with the PVF. Dialogue keys are used to protect subsequent operations between the client and the server. One of the keys is used to provide integrity protection, the other confidentiality (if required). They are different so that different strengths of protection can be provided.

2.3. Ticket and PAC Validation (No Delegation)

Protection of DCE 1.0 tickets continues to be supported unchanged. DCE 1.0 tickets carry access privileges inside them (the DCE 1.0 PAC) and remain non-delegatable. They are exchanged and protected via existing protocols and existing symmetric cryptographic techniques.

DCE 1.1 introduces a separation between the PAC (now extended to become the EPAC, as already described) and the DCE 1.1 ticket (which no longer contains privileges). The DCE 1.1 ticket itself is protected in the same way as the DCE 1.0 ticket, but the EPAC is not encrypted; it is integrity protected as described in Section 2.1.

Sealed non-delegatable EPACs are validated by the PVF by calculating the EPAC checksum and comparing it with the value found in the decrypted ticket.

Signed non-delegatable EPACs are validated by the PVF by:

(a) Calculating the EPAC checksum, decrypting the EPACs encrypted checksum using the public key of the PS, and comparing for equal.

(b) Ensuring that the TRID value in the EPAC is equal to the TRID value in the ticket (i.e., that the EPAC has come from the original requester of the PTGT in the ticket).

Both sealed and signed PACs can contain optional Target Qualifier Attribute fields (see also Section 3.2); the PVF will reject the EPAC if the target application at which the EPAC is aimed does not qualify. If the target application is a member of one of the groups listed, or its identity is listed, then the target qualifies.

2.4. Cryptographic Support

While existing DCE 1.0 security services will use encryption as before, a *Cryptographic Support Facility (CSF)* is provided for DCE 1.1 functions which allows plug-replaceable algorithms. The API to this is a general one providing interfaces to encrypt, decrypt, sign and verify data while hiding the actual algorithms and key formats used. These functions are implementations of:

(a) Symmetric cryptographic functions.

(b) Asymmetric cryptographic functions.

(c) Irreversible cryptographic (one way) functions.

(d) Key generation.

The CSF is a library used by security components such as the PS, PVF and RPC and components (such as the GSS-API support routines for non-RPC applications) which need to ensure the confidentiality and integrity of data in transit. The level of protection of the exchanges is configurable. It can, for example, provide weak confidentiality and strong integrity.

Public key technology is implemented as a natural extension to the cryptography facilities. The use of public key technology offers security benefits particularly in larger systems but is not mandatory, and cells supporting public key technology can use secret key technology to interwork with those which do not (though mixed technology delegation chains are not supported).

At DCE 1.1 public key technology is used only for signing EPACs. This reduces the number of exchanges with security servers needed for intercell working, since PS public keys can be directly known at servers in multiple domains without risk of compromise of the PS domain. It also improves the performance of delegation, since (as will be seen in Section 3) the PS does not need to be visited on each leg of the delegation chain.

Interfaces are put in place so no interface change is required when public key technology is used for key distribution.

Certificate Authorities (CAs) are used to sign public key certificates for PS public keys using the X.509 certificate syntax. Initially only the PS will use a public key pair, and servers will be able to obtain a certificate for its public key either from the CDS, or by manual installation in the target server system. CAs are expected to serve many cells, producing certificates for their PS's keys. Later, this will be extended to allow CAs to be arranged in a hierarchy, and to allow certificates to be obtained from the GDS, but in DCE 1.1 CA keys will be installed in servers by management action.

3. DELEGATION

3.1. Delegation Overview

A user or client may allow an application to use its rights by proxy, i.e., it may delegate its rights to another. This application may in turn delegate the client's rights further, for example, a user's rights may be delegated to a business application which then delegates them to a database server which uses them when handling records belonging to that user.

The client may want to control the extent to which its rights can be passed on. Each target application may wish to control access to its objects based on both the original user's rights and the delegates who passed them on.

The delegation proposal here is a unification of the existing DCE 1.1 delegation proposal [RFC 3.0], and the simpler CV/PV mechanism (see Section 3.3.1 for the definitions of CV and PV) originally proposed in the Sesame proposal dated 4th of July 1992. Some minor changes to [RFC 3.0] are proposed for compatibility with the asymmetric case and also some changes to the API are proposed to ease support of non-RPC as well as RPC applications.

Different levels of delegation are supported for DCE 1.1:

(a) *No delegation* — The client's PAC can only be used by its immediate target.

(b) *Simple delegation* — The client's rights can be delegated, and optionally constrained to identified targets or target groups. Delegates do not need to obtain EPACs of their own and thus no trace of the delegation route is provided for the target.

(c) *Traced delegation* — The client's rights can be delegated and optionally constrained to separately identifiable delegates and targets. The chain of delegates involved in a call from an initial client to the eventual target application is built up using the delegates' EPACs. Thus each delegate needs an EPAC. The attributes of the delegates as well as the original user/client can optionally be used by the target for access control decisions.

A target can upgrade to use fully controlled delegation facilities in gradual stages: first accepting simple delegation, then accepting delegation provided it is traced, then accepting delegation provided the traced delegates have appropriate access rights in the DACL. However, the simple level of delegation will be adequate for many cases.

All delegation options are provided for both symmetric and asymmetric EPAC protection options, with the same administration image and the same API support.

Applications at target systems can control access using the standard DCE ACLs with controls based on the original owner's rights. When tracing is used, a target can make access control decisions based not only on the access rights of the original initiator, but of all the delegates. For this case, the target modifies its DACL to include delegation access rights. The appropriate DACL semantics for this are described in [RFC 3.0].

While many applications are not expected to want detailed control over delegation, for those security aware applications which do want to set their own controls, APIs are provided as specified in Section 6.

3.2. Controlling the Use of EPACs

This Section describes the controls as seen by the administrator or security aware application which wants to control delegation. It is possible for delegation controls to be entirely specified by principals (humans and applications) calling on the APIs in Section 6. However, administrators will often want some control of delegation, in line with the security policy, particularly because many principals will not be sufficiently informed to do this themselves. The administrator's view is given in 5.2.

Delegation and other restrictions on the legitimate users of an EPAC are included in the EPAC itself, or passed with it as described below and can be checked by both delegates in the chain and the final target applications. Some delegates may use the client's rights in the EPAC for their own access control as well as passing them onto others.

The following control attribute types in the EPAC are used to validate whether a server can use the EPAC either as a target, or a delegate:

(a) *Target Qualifier Attributes (TQAs)* define the targets at which the EPAC is valid for control of access to that target's objects.

(b) *Delegate Qualifier Attributes (DQAs)* define which targets can use the EPAC as a delegate. Note therefore that all delegates are valid targets.

Applications can be grouped into ‘‘application trust groups’’ and the control attributes above can identify applications either individually or by their group membership. The advantage of the latter is the flexibility and ease of administration of delegation control. The administrator can choose what sort of groups he wants, depending on the installation's needs. For example, he might define a few large groups of applications to separate major areas such as administration, finance etc. Some services, such as file or print services, could belong to several groups. Alternatively, he could define more smaller groups for finer control. Tight controls have a penalty in the extra cost of implementing and administering them.

More generally, different levels of constraint can be provided by appropriately confined or open values when nominating valid targets or delegates. If no target controls are specified, an EPAC can be valid at any target which accepts that kind of delegation.

It is also possible to include other control attributes in the EPAC which restrict when and how it can be used. These include controls such as time constraints and attributes such as the ''required'' and ''optional'' restrictions described in [RFC 3.0]. Required and optional restrictions are represented by specific security attribute types in the EPAC and are supported for all forms of delegation.

A target can just insist that the route has been traced, without using delegate rights for access control. In this case, the target knows that all delegates have been empowered with the ability to become a delegate by virtue of the traceable EPACs they are able to obtain from the PS.

At the API, the client annotates a login context with delegation constraints. Subsequent calls to use this context cause a request to the PS for an EPAC containing attributes reflecting those constraints, and a request to the TGS for a ticket to talk to the server (where suitable ones have not been cached). The server receiving these items establishes a login context reflecting the delegation from the client. This server may annotate the context with further constraints. Again, suitable tickets and EPACs are forwarded to the next server called.

Note: Non-RPC applications will use the GSS-API style of interface and refer to ''credentials'' rather than login contexts. (In practice, the only difference between a login context and a set of credentials is the form of handle used to access them.)

3.3. Mechanisms

This Section describes the mechanisms used to support delegation in both symmetric and asymmetric EPAC options. These mechanisms are not visible at the API and this Section can be skipped on first reading by those who do not want to understand how the facilities are provided.

3.3.1. Protection values and control values

Protection and control values are used to protect EPACs, particularly when delegation uses asymmetrically signed EPACs. They allow the same EPAC to have different restrictions at different targets.

Any qualifier attributes about targets can be associated with a value in the EPAC known as a *Protection Value (PV)*. The PV is a one-way function of a matching value known as the *Control Value (CV)* which the PS returns to the client with the EPAC encrypted under the key being used to protect the EPAC request. (The PS calculates the PV from a randomly generated CV.) Apart from the PS itself, only the original requester of the EPAC now knows the CV, though subsequently other principals may learn it by being passed it with the EPAC.

The client sends the CV encrypted under the session key with the EPAC to the target. The target checks that:

(a) The CV corresponds to a PV in the EPAC.

(b) The target matches one of the qualifier attributes associated with this PV in the PAC.

The target now knows:

(a) The sender was either the original owner or has been given the CV by a principal that knows it.

(b) The sender, by passing this particular CV intends any controls associated with it to be checked. Note that if an EPAC contains a PV which is associated with no attributes of a particular type, then no checks of that type are made on an EPAC with which the corresponding CV is sent.

More than one group of controls can appear in an EPAC, each one associated with a different PV. When the same EPAC is used at different times with respect to different targets, an appropriate group of controls can be activated by the sender transmitting the corresponding CV.

Use of CV/PVs is not always required. For example they are not required for non-delegatable EPACs.

3.3.2. Symmetrically protected EPACs

Whether tracing is required or not, the mechanism for handling delegation is as described in [RFC 3.0] with the following differences:

(a) The syntax of each EPAC is the new one described in Section 8.

(b) The original EPAC contains an indication as to whether tracing is required or not.

(c) A CV can be sent with the EPAC to activate a nominated group of delegate and target qualifiers from the EPAC.

(d) Instead of building up a sequence of EPACs in the ticket, if tracing is required a sequence of check values is built up, accompanied by a corresponding sequence of EPACs, all integrity protected rather than encrypted.

(e) If tracing is not required, no intermediate EPACs are needed and only the original EPAC is presented at the final target.

3.3.3. Asymmetrically protected EPACs — simple delegation

The PVF at a target receiving an EPAC will first check whether it has come directly from the original requester of the EPAC. It does this by making the TRID check as described in Section 2.3. If this check fails, the EPAC is being delegated and must be accompanied by a CV. The target PVF checks that untraced delegation is acceptable to it, and if so, that the CV corresponds to a PV in the EPAC and that the DQA or TQA values associated with the PV are appropriate for itself. If these checks are passed, the EPAC can be said to have been properly presented by an acceptable delegate for use at this target.

The target itself is now potentially in a position to use the incoming EPAC acting as a delegate itself, since its PVF now knows the CV. However, the PVF will only release the CV if the target is a valid delegate.

In simple delegation, delegates do not require EPACs of their own, and do not need to visit the PS to get their position as delegates authorised in any way.

3.3.4. Asymmetrically protected EPACs — traced delegation

When public key technology is being used to provide EPAC protection, delegation with trace is provided using EPAC chaining along the lines of [VAB]. The provision of DQA controls in the initial EPAC in the chain prevents the delegation attack to which [VAB] indicates that this method is susceptible. The chaining technique is also more flexible: multiple potential forward pointers can be present in an EPAC, from which one can be dynamically selected on any particular use of the EPAC by using a CV as described in Section 3.3.1. The additional EPAC fields that have been defined for chaining are:

(a) *Next Target Qualifier Attribute (NTQA)*. This provides a link to the next EPAC in a delegation chain. It is the same type of attribute as the target and delegate qualifier attributes; in DCE 1.1, it is the identity of the owner of the next EPAC in the chain.

This field is associated with a flag which indicates if this PAC is of a delegate or not.

(b) *Owner Qualifier Attribute (OQA)*. This identifies the principal which is the original owner of the EPAC containing it. Note that this attribute is not used in access control decisions. It is used to provide a link to the previous EPAC in a delegation chain using its NTQA.

The NTQA in an EPAC in a delegation chain will have the same value as the OQA of the next EPAC in the chain.

When delegation is traced, delegates require EPACs. However, provided that the delegates on the route have already obtained EPACs containing appropriate NTQAs there is no need for a delegate to visit the PS for each instance of an actual delegation (delegate EPACs can be relatively long lived and they can for example be obtained at the start of day — they can be created with NTQAs directed at commonly used next targets).

A sequence of EPACs is built up along the delegation route in a similar way to the symmetric case. The ticket accompanying them however contains, on each leg, only the TRID of the direct invoker; i.e., the ticket is always a normal ticket for the last EPAC that was added and no accumulation of TRIDs is made in tickets. This permits tickets that have been already obtained by delegates to be used with newly received EPACs, with consequent performance improvements.

A target receiving such a sequence with a ticket makes the following checks:

(a) The first EPAC in the sequence is not a Delegate EPAC. (See description of NTQA above.)

(b) For each EPAC in the chain, it is targeted at an identifiable single next target, i.e., it either contains only one NTQA control, or it is accompanied by a CV which selects one NTQA control.

(c) For each EPAC in the chain, the chain is intact, i.e., next EPAC in the sequence contains an OQA which matches the preceding EPAC's NTQA.

(d) The last EPAC in the chain has come directly from its original requester, i.e., it satisfies the TRID check (see Section 2.3).

(e) All the intermediate EPACs in the chain are from valid delegates, i.e., the OQAs in the delegates' EPACs satisfy the applicable DQA controls of the EPACs preceding them.

(f) The target itself satisfies the applicable TQA or DQA controls of all of the EPACs received.

4. HANDLING EXTENDED ATTRIBUTES

This proposal supports an extended range of attribute types and values in EPACs as follows:

(a) *Privilege attributes* associated with a user or other principal.

(b) An *audit identity* as described in Section 5.1.

(c) *Control attributes* such as target and delegate qualifiers and required and optional restrictions as described in Section 3.

The APIs at target systems permit these attributes to be extracted in a transparent way so that API callers there (often the applications themselves) can use them in access control decisions independently of any local operating system controls. In this way the door will be opened to OSF DCE users for them to provide further extensions of their own design to the support of access control policy in applications. These can either be entirely within the OSF DCE regime, or to support interworking with non-DCE systems.

4.1. Privilege Attribute Types Supported

[RFC 6.0] provides registry support for an extended range of attribute types and values. To complement this, the syntax of the EPAC has been designed so that any of these attributes can be carried in it. The syntax can be anything [RFC 6.0] permits, and therefore does not need to be confined to the current DCE/UNIX style attributes such as UUIDs.

DCE 1.1 continues to supports the current principal identity and group attributes and also supports a role attribute which is a type of group attribute. All these are in UUID form and can be used in existing DCE ACLs.

Attributes can be specified with values other than UUIDs. For example, global names, directory distinguished names, simple strings or ASN.1 object identifiers can be specified. The attribute can have a pair of values with a defining authority permitting attributes other than those defined in the initiating cell to be inserted in the EPAC.

4.2. Roles and Role Names

The user wants a simple interface to the system so does not want to have to understand and request all the privileges and controls he needs. The administrator wants to control what privileges the user is allowed but does not want to specify each privilege individually for each user and then update these whenever a user's job changes. Similarly, many business applications would like to control access on the basis of the job an individual is doing in the organisation. The user must be uniquely accountable for his or her actions, but the user's identity need play no part in actual access control decisions. In this way, access control lists can be established independently of knowing which specific users have what particular jobs in the organisation, and are insulated from management change when the user population changes.

To help satisfy these requirements, the concept of a role is introduced. A user may potentially be allowed to take more than one role, though will be acting in only one at a time. Not all users need have roles, since in some security policies, roles are not relevant.

A user working in a particular role is likely to need particular privileges and controls which give the required access to particular applications and services. For example, he may need to be a member of a particular group and have access to particular target application groups.

The administrator defines the set of attributes which is associated with a role. He also defines which users can take which roles. At the start of a session, the user specifies the role name (or gets a default) which automatically gives him the associated attributes.

Some of the advantages of this are:

(a) The user only needs to know the role name (e.g., administrator, marketeer), not what attributes this needs. If he always works in the same role, he doesn't even need to know this, as it will be his default role.

(b) If the user changes job, once the administrator has updated his roles, when he next uses the system, he automatically gets the privileges associated with the new job.

(c) Administration is significantly simplified if many users take the same role, as the set of attributes for the role only need defining once. Also, some delegation controls can be associated with roles rather than individual applications.

(d) If a role attribute (or other group) is associated with the job role, access control in some types of application can be based on this, rather than on individual user identities, thus simplifying administration of the target system. (A role attribute is recommended as being obviously associated with the role — the existing group concept is already rather overloaded.) Note that using the audit identity in the EPAC, the target system can still provide audit trails in which the user is individually accountable, without having to be concerned about managing that value.

Support of roles requires some extra information in the registry which the administrator can maintain. This is:

(a) The roles and their associated attributes.

(b) A list of permitted roles associated with a principal, with a default per user and also a system default when there is no role associated with a user.

This is done using the extended registry attribute proposal ([RFC 6.0]) with some additions.

4.3. Using the Extended Attributes

The extended attributes are defined and used as follows:

(a) The extended attributes associated with a principal or role will be defined in the registry using the extended registry facilities proposed in [RFC 6.0].

(b) The user/client optionally specifies what attributes are required either for the complete session or when calling a particular application. When the PS is called to obtain an EPAC, the extended PS checks the client's rights to these privileges before including them in the PAC.

(c) At the target system, the server optionally calls an RPC or extended GSS-API ‘‘get attributes’’ interface to request information about the security attributes in the appropriate security context associated with the client.

(d) The server may then use these for access control either by using the DCE ACL Manager (if they are in UUID form) or its own authorisation facility.

In DCE 1.1 targets, the DACL manager routines will be modified to use the ‘‘get attributes’’ interface to obtain the incoming context attributes instead of the current interface which is specific to DCE 1.0 PACs. This provides access to the extended attributes, such as the delegation controls, and will also enable the DACL routines to be insulated from any further changes to the methods by which a principal's privileges are established.

4.4. Handling Attributes Across Cells

For attributes like principal ids and groups, which are identified by separate UUIDs for the cell and principal/group, existing DCE 1.0 rules are normally used. Only attributes from the client cell are accepted and can be used for access control in DCE ACLs and elsewhere.

However, DCE 1.1 allows an administrator to set up trust relationships with specified cells (or all cells) and decide whether to accept all or just certain attribute types from these cells. This is not confined to attributes with UUID values, it can apply to attributes types with, for example, global names or defining authority and value.

Some attribute types are expected to be understood across cells, for example the application trust group used in delegation.

5. OTHER EXTENSIONS

5.1. Auditing

All security related events detected by the enhanced DCE 1.1 security components can be audited. A unique *audit ID* is assigned to each principal to ensure personal accountability across the system. It is an extended attribute of a user and is put in the EPAC by the PS.

The audit ID of the initiator, and those of the delegates if delegate tracing is in use, are extracted from the EPACs and written to the audit records.

The following events are audited by components described in this DCE-RFC:

(a) Login — by the extended DCE login mechanism.

(b) Get privileges/refresh PTGT — by the Privilege Server.

(c) Start/terminate/failure of a secure association — by the security runtime.

(d) PAC validation and key distribution — by the PAC Validation Facility.

Furthermore, all security relevant administrative operations can be audited.

This audit proposal fits with [RFC 25.0].

5.2. Administration

The administrator can set delegation and other controls as follows:

(a) Set major security policy options such as:

   (i) All delegation is to be traced.

   (ii) Default integrity and confidentiality levels.

   — so that these do not need to be specified by all applications at run-time.

(b) Decide on the level of control of delegation targets needed and set up suitable application groups to support this. When an application is introduced to the system, the administrator will define what group(s) it belongs to.

(c) Where user roles are used (see Section 4.2), decide what delegation constraints should apply to which roles.

(d) Where delegation may span cells, agree with the administrator of the foreign cell on any attributes, such as application groups, which both cells should understand and trust each other to put in EPACs only of suitable principals.

5.3. Other Supporting Changes

Other changes are associated with facilities described earlier in the document.

(a) Changes to RPC run-time to utilise the new facilities.

(b) Changes to the DACL Manager to get attributes from the EPAC (also needed for [RFC 3.0]).

(c) Changes to dce login to support the extended facilities.

(d) Extensions to the registry schema to add the extended attributes and roles described in 3 and 4 above.

(e) Optional extensions to the information held about applications in the name server to identify what type of ticket/EPAC the application can accept.

6. APIS FOR ATTRIBUTE REQUEST, RETRIEVAL AND DELEGATION

This Section specifies the APIs used by applications. These are generic APIs which hide the details of the security mechanisms used. For RPC applications, they extend the existing RPC API so use rpc binding handles. For non-RPC applications, the generic APIs are an extension to the GSS-API as proposed in [RFC 5.0]. Underlying these are the interfaces to the security services such as the enhanced Privilege Server. The PS interface is only used by security software such as that supporting the RPC run-time library and is described in Section 7.

6.1. RPC Interfaces

6.1.1. rpc_binding_set_attributes()

This operation annotates the specified binding handle so that it can be used to make authenticated RPC calls. Used by the more security aware RPC applications, it sets the controls that are to be applied (e.g., validity time, restrictions on attributes, proxy) in addition to those already associated with the binding handle. The authentication and authorisation information for the binding handle is also set up, see `rpc_binding_set_auth_info(3rpc)`. Unless an application calls this routine or `rpc_binding_set_auth_info(3rpc)` all remote procedure calls made on the binding handle are unauthenticated. This routine may be used instead of `rpc_binding_set_auth_info(3rpc)` by security aware applications.

Signature:

```
void
rpc_binding_set_attributes (
[in]  rpc_binding_handle_t    binding,
[in]  unsigned_char_t         *princ_name,
[in]  rpc_auth_identity       auth_identity,
[in]  rpc_auth_info_t         auth_info,
[in]  gss_attribute_set       *privilege_attrib_values,
[in]  gss_target_control_set  *target_attrib_values,
[out] unsigned32              *status );
```

6.1.2. rpc_binding_inq_attributes()

This operation gets the authentication attributes associated with the specified binding handle. The calling client associates the authentication attributes with the server binding handle by prior use of the set authentication binding attributes operation. The RPC runtime allocates memory for the returned `princ_name` argument. The caller is responsible for calling the `rpc_string_free()` routine for the returned argument string. This routine may be used instead of `rpc_binding_inq_auth_info(3rpc)` by security aware client applications. It returns the controls that are applied (e.g., validity time, restrictions on attributes, proxy). The authentication and authorization information from the binding handle is also returned (see `rpc_binding_inq_auth_info(3rpc)`).

Signature:

```
void
rpc_binding_inq_attributes (
[in]  rpc_binding_handle_t    binding,
[out] unsigned_char_t         *princ_name,
[out] rpc_auth_info_t         *auth_info,
[out] gss_attribute_set       *privilege_attrib_values,
[out] gss_target_control_set  *target_attrib_values,
[out] unsigned32              *status );
```

6.1.3. rpc_binding_set_proxy_info()

A server manager routine uses this operation to set-up a server binding handle so that it can be used to make authenticated remote procedure calls on behalf of the client to another server. It sets up a binding handle for an outgoing RPC on behalf of the distant client. Used by server applications.

Signature:

```
void
rpc_binding_set_proxy_info (
[in]  unsigned_char_t         *other_server_princ_name,
[in]  rpc_binding_handle_t    client_binding_handle,
[in]  rpc_binding_handle_t    other_server_binding_handle,
[in]  gss_target_control_set  *target_attrib_values,
```

```
[out] unsigned32             *status );
```

6.1.4. rpc_binding_inq_attrib_client()

Returns the security attributes from the binding handle for an authenticated client. This operation gets the authentication attributes associated with the client specified client binding handle. The calling server manager routine can use the returned data for authorization purposes. The authentication and authorization information from the server binding handle is also returned (see `rpc_binding_inq_auth_client(3rpc)`).

Signature:

```
void
rpc_binding_inq_attrib_client (
[in]  rpc_binding_handle_t    binding,
[out] rpc_auth_info_t         *auth_info,
[out] gss_attribute_set       *privilege_attrib_values,
[out] unsigned32              *status );
```

6.2. Generic APIs

For non-RPC applications, the Generic Security Service API defined in [GSS-API] is used for supporting secure associations between client and server and support facilities such as mutual authentication and data confidentiality and integrity. The most important GSS-API calls are `GSS_Init_sec_context()` which is called by a client (using the default or specified credentials) to obtain the security information needed and make the association, and `GSS_Accept_sec_context()` which is called by the server accepting the association.

The calls to change between DCE login contexts and GSS-API credentials and vice versa (`GSSDCE_LoginContext_To_Cred()` and `GSSDCE_Cred_To_LoginContext()`) specified in [RFC 5.0] are also supported.

In addition to these, two extra calls are introduced. *Modify credentials* is used to request a set of privileges and controls and *Get attributes* allows the caller to retrieve attributes associated with credentials.

6.2.1. Modify credentials

This is used to request a set of privileges and controls, optionally modifying existing credentials or creating a new set. Repeated calls to modify an existing set of credentials are cumulative.

Signature:

```
OM_uint32
gss_modify_cred (
[in]  gss_cred_id_t           cred_han,
[in]  OM_uint32               duplicate_cred_req,
[in]  gss_attribute_set       required_privilege_attributes,
[in]  gss_target_control_set  required_target_controls,
[out] OM_uint32               *minor_status,
[out] gss_cred_id_t           *output_cred_han,
[out] gss_attribute_set       actual_privilege_attributes,
[out] gss_target_control_set  actual_target_controls );
```

6.2.2. Get attributes

This is used to get attributes associated with credentials or security context. Two important usages of this API are:

(a) Audit-id retrieval.

(b) Use by ACL manager (instead of `rpc_binding_inq_auth_client()`) to get privileges.

Signature:

```
OM_uint32
gss_get_attributes (
[in]  gss_cred_id_t           cred_handle,
```

```
[in]  gss_ctx_id_t            context_handle,
[in]  gss_id_set              attribute_types_required,
[out] OM_uint32               *minor_status,
[out] gss_attribute_set       attributes,
[out] gss_target_control_set  actual_target_controls );
```

6.3. Supporting Types for APIs

```
typedef {
   gss_type_en     id_type;
   gss_value       id_value;
}  gss_id;

typedef enum {
   gss_oid_t,
   gss_integer,
   gss_string,
   gss_uuid,
   gss_buffer
}  gss_type_en;

typedef struct union {
   gss_OID         OID;
   OM_uint32       *integer;
   char            *string;
   uuid_t          *uuid;
   gss_buffer_t    buffer;
}  gss_value;

typedef struct {
   OM_uint32       value_count;
   gss_value       *value_list;
}  gss_value_list;

typedef struct gss_attribute_desc {
   gss_id          attribute_type;
   gss_id          policy_authority;
   gss_type        value_syntax;
   OM_uint32       values_count;
   gss_value_list  *values_list;
}  gss_attribute;

typedef struct gss_id_set_desc {
   OM_uint32       id_count;
   gss_id          *ids;
}  gss_id_set;

typedef struct gss_attribute_set_desc {
   OM_uint32       attribute_count;
   gss_attribute   *attributes;
}  gss_attribute_set;

typedef struct {
   OM_uint32       time_count;
   time_t          *time_list;
}  gss_time_list;
```

```
typedef struct {
   OM_uint32       control_count;
   gss_controls    *control_list;
}  gss_control_set;

typedef struct gss_controls_desc {
   OM_uint32       *check_back;
   OM_uint32       *usage_count;
}  gss_controls;

typedef struct target_control_desc {
   gss_attribute_set  *target_attributes;
          /* Attributes characterising a group of targets.   */
          /* If GSS_C_NULL_ATTRIBUTE_SET is specified,       */
          /* then all targets are included.                  */

   gss_control_set   *controls_required;
          /* Controls to be applied.                         */

   OM_uint32         delegation_flags;
          /* Sensible combinations of the following flags    */
          /* describe "target_attributes".                   */
          /* Values supported at DCE 1.1 are:                */
          /* - none:            no delegation permitted      */
          /* - can_be_delegate: can act as delegate & target */
          /* - trace_required:  must be traced               */
} target_control;

typedef struct {
   gss_time_list          valid_time_periods;
   OM_uint32              target_control_count;
   target_control         *target_controls;
}  gss_target_control_set;
```

6.3.1. Example

Suppose the required controls are:

(a) T1 and T2 are valid targets, but cannot be delegates;

(b) T4 may be a delegate and a target;

(c) trace is required for delegation via T6.

Then the `gss_target_control_set` structure is as follows (in pseudo-code):

(a) `target_attributes` = T1, T2;

(b) `delegation_flags` = none;

(c) `target_attributes` = T4;

(d) `delegation_required` = `can_be_delegate`;

(e) `target_attributes` = T6;

(f) `delegation_required` = `can_be_delegate | trace_required`.

7. PRIVILEGE SERVER PROTOCOL ENHANCEMENT

The following are the RPC calls used to obtain PTGTs and EPACs from the Privilege Server. These calls are not normally visible to applications who use the interfaces described in Section 6. They can be used by clients in a non-DCE environment. The data structures used are described in Section 8. New parameters for DCE 1.1 are noted.

7.1. rpriv_get_ptgt_epac()

Signature:

```
void
rpriv_get_ptgt_epac (
[in]       handle_t                handle,
[in]       unsigned32              authn_svc,
[in]       unsigned32              authz_svc,
[in]       rpriv_pickle_t          ptgt_req,
[in]       boolean                 new_ptgt_needed,  /*NEW*/
[in]       rpriv_pickle_t          existing_ptgt,  /*NEW*/
[in]       unsigned32              type_pac,  /*NEW*/
[in]       unsigned_char_t         role_name,  /*NEW*/
[in]       gss_attributes_set      req_privileges_attributes,
                                   /*NEW*/
[in]       gss_target_control_set  required_target_controls,
                                   /*NEW*/
[out]      rpriv_pickle_t          ptgt_rep,
[out]      pac_t                   extended_pac,  /*NEW*/
[out]      c_value_set_t           list_cv,  /*NEW*/
[out,ref]  error_status_t          status );
```

Input:

(a) `handle`, `authn_svc`, `authz_svc`, `ptgt_req` — As in DCE 1.0.

(b) `new_ptgt_needed` —

  (i) `TRUE` if the PTGT is required (in first call for example).

  (ii) `FALSE` just a new EPAC is required. The TRID is taken from existing PTGT.

(c) `existing_ptgt` — Only used to obtain a new EPAC linked to an existing PTGT. This field provides the value of the TRID to put in the EPAC.

(d) `type_pac` — The type of PTGT the client wishes:

  (i) `SEC_DCE1_0` — for DCE 1.0 PTGT.

  (ii) `SEC_DCE_SYM` — for DCE 1.1 PTGT using symmetric key.

  (iii) `SEC_DCE_ASYM` — for DCE 1.1 PTGT using asymmetric key.

(e) `role_name` — Specifies attribute required for this PTGT. If `NULL`, the Privilege Server assumes a default value. This argument is only valid for EPAC.

(f) `req_privileges_attributes` — A set of privileges required in the PAC. This list is added to the privileges for the required role.

(g) `required_target_controls` — A set of target attributes to control the required EPAC (for example by use in delegation).

Output:

(a) `ptgt_rep` — If the request is valid and if the PTGT is required, a DCE 1.0 protocol structure with the PTGT depending on the `type_pac` value supplied in input. A EPAC may be linked with this PTGT.

(b) `extended_pac` — If the request is valid, the extended PAC signed or sealed under the required method. If a DCE 1.0 PTGT is wished, this argument is `NULL`.

(c) `list_cv` — A structured list of control values for this EPAC. This list is `NULL` for `SEC_DCE1_0` type. It is returned to the client using DCE 1.0 session encryption protection.

7.2. rpriv_refresh_ptgt()

Signature:

```
void
rpriv_refresh_ptgt (
[in]       handle_t          handle,
[in]       unsigned32        authn_svc,
[in]       unsigned32        authz_svc,
[in]       rpriv_pickle_t    service_ticket,
[in,out]   rpriv_pickle_t    ptgt_arg,
[out,ref]  error_status_t    status );
```

Input:

(a) `handle` — The context handle.

(b) `service_ticket` — The PS service ticket which supplies the session key.

(c) `ptgt_arg` — The expired PTGT with the authorization data to be skipped and the renew time for this ticket.

Output:

(a) `status` — A returned value.

(b) `ptgt_arg` — The new PTGT with the authorization data (i.e., asymmetric or symmetric case) supplied by the `ptgt_arg` referred by the input parameter. The different data structure are encrypted as before. The PS modifies the expired time in the PTGT. The validity of the ticket is limited by the ‘‘renew-till’’ of the PTGT.

Description: The PS modifies the validity time as defined in the registry policy, checks the validity time for renew PTGT and signs against the EPAC.

8. DATA STRUCTURES

This DCE-RFC introduces:

(a) Changes to the authorisation data field of tickets such as PTGTs.

(b) The EPAC described in Section 2.

(c) A *dialogue key package* exchanged between client and server to enable dialogue keys to be established.

The data is described here in DCE IDL, though an ASN.1 definition of the PAC is also available. This Section describes the essential components, not all fields are expanded here.

8.1. Ticket Authorisation Data

This field of the ticket contains the cryptography technology used to protect the PAC; and, for the symmetric cryptography, the PAC serial number and a hash of the PAC.

```
typedef struct {
     sec_id_format_t pac_type;  /* sym or asym EPAC          */
     pac_desc_t pac_desc;
} authorization_data;

typedef enum {
     sec_id_format_v1,          /* a DCE 1.0 PAC             */
```

```
     sec_id_format_v1_1_asym,   /* a DCE 1.1 asymmetric PAC  */
     sec_id_format_v1_1_sym     /* a DCE 1.1 symmetric PAC   */
} sec_id_format_t;

typedef union switch (sec_id_format_t pac_type) tagged_union {
     case sec_id_pac_format_v1:
       /* TBD */;               /* DCE 1.0 PAC type          */
     case sec_id_pac_format_v1_1_asym:
       trid_t trid;             /* the TRID included in PAC  */
     case sec_id_pac_format_v1_1_sym:
       octet_string_t pac_hash; /* result of the hash of PAC */
     case default:
       ;                        /* empty branch of union     */
} pac_desc_t;
```

8.2. PAC Syntax

```
typedef struct {
    certificate_id_t     certificate_id;
                                /* identification of the PAC */
    utc_time_t           creation_time;
                                /* time of creation          */
    validity_time_t      time_period;
                                /* validity time period      */
    unsigned16           method_group_num;
                                /* control and PAC protection*/
    [size_is(method_group_num),ptr]
        method_group_t   *method_groups
    security_attribute_t owner_qualifier;
                                /* PAC owner                 */
    pac_type_t           pac_type;
                                /* type of the PAC           */
    unsigned8            pac_syntax_version;
    restrictions_t       restrictions;
                                /* opt'l & req'd restrictions*/
    unsigned16           priv_num;
                                /* list of the privileges    */
    [size_is(priv_num),ptr]
         privilege_t     privileges;
                                /* user & group UUIDs, ...   */
                                /* others (customer defined) */
    unsigned16           identifier_num;
                                /* list of the identifiers   */
    [size_is(identifier_num),ptr;
                                /* authenticated identity    */
        identifier_t     *identifiers;
                                /* audit identity            */

    check_value_t        check_value;
                                /* cryptographic check value */
                                /* (asymmetric signature     */
                                /* or symmetric seal)        */
} pac_t;

/* ------------ certificate id ----------------- */
```

```
typedef struct {
    sec_id_t             issuer_domain;
                                /* cell uuid                  */
    sec_id_t             issuer_identity;
                                /* PS uuid                    */
    unsigned32           serial_number;
} certificate_id_t;

/* ------------ validity time ------------------ */

typedef utc_time_t       unsigned32;

typedef struct {
    utc_time_t           start_time;
    utc_time_t           end_time;
} validity_time_t;

/* ------------ protection methods ------------- */

/* method_group_t is a sequence of protection_t. */

typedef struct {
    protection_type_t    protection_type;
                                /* Type of protection of PAC */
    unsigned16           num_param;
    [size_is(num_param),ptr] protection_param_t
                         protection param;
                                /* Parameter corresponding to*/
                                /* the protection method type*/
} protection_t;

typedef enum {
    trid,                       /* TRID method for            */
                                /* non-delegatable PACs       */
    external_hash,              /* for symmetric key only;    */
                                /* indicates that the hash of*/
                                /* the EPAC is in the ticket */
    control_protection_value,   /* CV/PV delegation method    */
    delegate_qualification,
    target_qualification,
    next_target,                /* Next target for traceable */
                                /* asymmetric delegation      */
} protection_type_t;
```

8.3. Dialogue Key Package

```
typedef struct {
    algorithm_identifier_t   owf_alg_id;
                                /* one-way function ID        */
    key_seed_t      integrity_key_seed;
                                /* seed to obtain integrity  */
                                /* key from session key       */
    unsigned16      integrity_key_size;
                                /* cut off of the result of  */
                                /* owf(session_key,           */
                                /*     integrity_key_seed)    */
```

```
        key_seed_t      confidentiality_key_seed;
        unsigned16      confidentiality_key_size;
    } dialogue_key_package_t;
```

9. ACKNOWLEDGEMENTS

This DCE-RFC has been produced by the Sesame consortium, with the consultation and encouragement of Joseph Pato (HP). Sesame members who contributed to this document include Eric Baize (Bull), Hiep Doan (Bull), Belinda Fairthorne (ICL), Stephen Farrell (SNI), Peter Hartmann (SNI), Per Kaiser (SNI), Piers McMahon (ICL), Tom Parker (ICL) and Denis Pinkas (Bull).

REFERENCES

[RFC 1.0] D. Nicely, ‘‘POSIX/C2 Auditing of DCE API's’’, June 1992.

[RFC 3.0] J. Pato, ‘‘Extending the DCE Authorization Model to Support Practical Delegation’’, June 1992.

[RFC 5.0] J. Linn, ‘‘GSS-API Extensions for DCE’’, June 1992.

[RFC 6.0] J. Pato, ‘‘A Generic Interface for Extended Registry Attributes’’, June 1992.

[RFC 7.0] J. Pato, ‘‘Hierarchical Trust Relationships for Inter-Cell Authentication’’, July 1992.

[RFC 25.0] E. McDermott, ‘‘DCE Auditing Design and Strategy’’, December 1992.

[VAB] V. Varadharajan, P. Allen, S. Black, ‘‘An Analysis of the Proxy Problem in Distributed Systems’’, Hewlett Packard Laboratories, Bristol, UK.

AUTHOR'S ADDRESS

Belinda Fairthorne
ICL
Kings House
33 Kings Road
Reading
Berkshire RG1 3PX
ENGLAND

Internet email:
s.b.fairthorne@rea0805.wins.icl.co.uk
Telephone: +44 734 855127