Network Working Group
Request for Comments: 3767
Category: Standards Track

S. Farrell, Ed.
Trinity College Dublin

June 2004

Securely Available Credentials Protocol

Status of this Memo

Copyright Notice

Abstract

This document describes a protocol whereby a user can acquire cryptographic credentials (e.g., private keys, PKCS #15 structures) from a credential server, using a workstation that has locally trusted software installed, but with no user-specific configuration. The protocol's payloads are described in XML. This memo also specifies a Blocks Extensible Exchange Protocol (BEEP) profile of the protocol. Security requirements are met by mandating support for TLS and/or DIGEST-MD5 (through BEEP).

Table Of Contents

1. Introduction

Digital credentials, such as private keys and corresponding certificates, are used to support various Internet protocols, e.g. S/MIME, IPSec, and TLS. In a number of environments, end users wish to use the same credentials on different end-user devices. In a "typical" desktop environment, the user already has many tools available to allow import/export of these credentials. However, this is not very practical. In addition, with some devices, especially wireless and other more constrained devices, the tools required simply do not exist.

This document describes a protocol for the secure exchange of such credentials and is a realization of the abstract protocol framework described in [RFC3760].

Many user-chosen passwords are vulnerable to dictionary attacks. So the SACRED protocol is designed to give no information with which an attacker can acquire information for launching a dictionary attack, whether by eavesdropping or by impersonating either the client or server.

The protocol also allows a user to create or delete an account, change her account password and/or credentials, and upload the new values to the server. The protocol ensures that only someone that knew the old account password is able to modify the credentials as stored on the credential server. The protocol does not preclude configuring a server to disallow some operations (e.g. credential upload) for some users. The account management operations as a whole are optional implementations for both credential servers and clients.

Note that there are potentially two "passwords" involved when using this protocol - the first used to authenticate the user to the credential server, and the second to decrypt (parts of) the credential following a download operation. Where the context requires it, we refer to the former as the account password and the latter as the credential password.

Using a protocol such as this is somewhat less secure than using a smart card, but can be used until smart cards and smart card readers on workstations become ubiquitous, and can be useful even after smart cards are ubiquitous, as a backup strategy when a user’s smart card is lost or malfunctioning.

The protocol sets out to meet the requirements in [REQS]. Cryptographic credentials may take the form of private keys, PKCS #15 [PKCS15], or structures. As stated, a profile based on BEEP [BEEP] is specified for message transport and security (integrity,

authentication, and confidentiality). In that case, the security requirements are met by mandating support (via BEEP) for TLS [TLS] and/or DIGEST-MD5 [DIGEST-MD5].

We assume the only authentication information available to the user is a username and password.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. The Protocol

This section defines the account management and "run-time" operations for the SACRED protocol.

It also describes the message formats used, which are described in XML [XMLSCHEMA]. Appendix A provides an XML schema for these elements.

The approach taken here is to define SACRED elements that are compatible with the elements used in [XKMS] and [XMLDSIG], so that an implementation of this protocol can easily also support XKMS, and vice versa.

It is also intended that other SACRED protocol instances (e.g. using a different authentication scheme, credential format, or transport protocol) could re-use many of the definitions here.

2.1. Account Management Operations

These operations MAY be implemented, that is, they are OPTIONAL.

2.1.1. Information Request

This operation does NOT REQUIRE authentication.

The purpose of this operation is to provide the client with the values required for account creation.

The client sends an InfoRequest message (which has no content).

The server responds with an InfoResponse message which contains the authentication mechanism parameters for the server and the list of supported ProcessInfo types. For DIGEST-MD5, this consists of the list of realms (each as an XML element named "Realm") which the server supports. There MUST be at least one realm specified.

Clients MUST be able to select one from a list of Realms and MUST be able to disregard any other information present (allowed for extensibility).

2.1.2. Create Account

This operation REQUIRES server authentication.

The purpose of this operation is to setup a new account on the server. The information required for a "new" account will depend on the SASL [SASL] mechanism used.

The client sends a CreateAccountRequest, which contains the account name (e.g. username). It also contains the elements required to create an account for a particular authentication mechanism. The actual information is defined according to the authentication mechanism. For DIGEST-MD5, this consists of the password verifier (the hashed username, password and realm) and the chosen realm. Although more than one set of such data is allowed by the data structures defined in the appendix, clients SHOULD only include one here.

The server responds with an error or acknowledgement message.

2.1.3. Remove Account

This operation REQUIRES mutual authentication.

The purpose of this operation is to delete the entire account.

The client sends a RemoveAccountRequest message (which has no content) to the server.

The server MUST delete all information relating to the account and respond with an error or acknowledgement message.

2.1.4. Modify Account

This operation REQUIRES mutual authentication.

The purpose of this operation is to allow the client to change the information required for authentication. The information required will depend on the authentication method used.

The client sends a ModifyAccountRequest message, which contains the elements required to change the authentication information for the account, for a particular authentication mechanism. The actual information is defined according to the authentication mechanism. For [DIGEST-MD5], it will consist of a realm and password verifier value.

Once the account information has been changed, the server will respond with an error or acknowledgement message.

2.2. "Run-time" Operations

These operations MUST be supported by all conformant implementations.

2.2.1. Credential Upload

This operation REQUIRES mutual authentication.

The purpose of this operation is to allow the client to deposit a credential with the server.

The client sends an UploadRequest message to the server which MUST contain one Credential.

If a credential with the same credential selector field as in the UploadRequest (a "matching" credential) already exists for the account, then that credential is replaced with the new credential from the UploadRequest. Otherwise a "new" credential is associated with that account. If a new credential is being uploaded, then the client SHOULD include (in LastModified) its local concept of the time (if it has one), or an indicator that it has no clock. The actual value of LastModified can be anything, (but the element has to be present) since this will be overwritten by the server in any case.

If any change is made to the stored credentials associated with the account, then the server MUST update the corresponding LastModified value (returned in DownloadResponse messages) to the current time (at the server).

The LastModified value in the UploadRequest MUST be the value which was most recently received in a corresponding DownloadResponse for that credential. This means the clients are strongly RECOMMENDED to only produce an UploadRequest based on recently downloaded credentials, since otherwise the LastModified value may be out of date.

The LastModified value can also be of use in detecting conflicts. For example, download to platform A, download to platform B, update from B, update from A. The server could detect a conflict on the second upload. In this case the server MUST respond with a BEEP error (which SHOULD be StaleCredential).

The server replaces the provided LastModified value with the current time at the server before storing the credential. (Note that this means that it would be unwise for a client to include the LastModified field in a ClientInfo digital signature which is calculated over the CredentialType.)

The server responds with an error or acknowledgement message.

2.2.2. Credential Download

This operation REQUIRES mutual authentication.

The purpose of this operation is to allow a client to get one or more credentials from a server (the purpose of the entire protocol really!).

The client sends a DownloadRequest message to the server which MAY contain a credential selector string for the credential. No, or an empty credential selector means the request is for all credentials associated with the account.

The server responds with a DownloadResponse or an error message. A DownloadResponse contains one or more credential payloads, including the LastModified time which represents the time (at the server) when the last change was made to each credential associated with the account (e.g. subsequent to an UploadRequest).

2.2.3. Credential Delete

This operation REQUIRES mutual authentication.

The purpose of this operation is to allow the client to delete one or all credentials associated with the account.

The client sends a DeleteRequest message to the server which can contain either a CredentialSelector or an All element.

If the DeleteRequest contains an All element, then all of the credentials associated with that account are deleted.

If the DeleteRequest contains a CredentialSelector, then the request MAY include a LastModified value. If the LastModified value is present in the DeleteRequest, then it MUST be the value which was most recently received in a corresponding DownloadResponse for that credential. If the value does not match, then the server MUST NOT delete the credentials.

If no "matching" credential exists, the server returns an error.

The server responds to this request with an error or acknowledgement message.

2.3. Miscellaneous

2.3.1. Session Security

Six SACRED operations are defined above. In this section we specify the requirements for security for each of the operations (where supported).

Operation	Security REQUIRED
Information request	NONE
Create account	Server authentication, Confidentiality, Integrity
Remove account	Mutual authentication, Confidentiality, Integrity
Modify account	Mutual authentication, Confidentiality, Integrity
Credential upload	Mutual authentication, Confidentiality, Integrity
Credential download	Mutual authentication, Confidentiality, Integrity
Credential delete	Mutual authentication, Confidentiality, Integrity

The security requirements can be met by several mechanisms. This document REQUIRES credential servers to support TLS and DIGEST-MD5. Clients MUST support DIGEST-MD5 and TLS with server authentication.

The mandatory-to-implement TLS cipher suite for SACRED is TLS_RSA_WITH_3DES-EDE_CBC_SHA. Implementations SHOULD also support TLS_RSA_WITH_AES_128_CBC_SHA [TLSAES].

When performing mutual authentication using DIGEST-MD5 for the client, DIGEST-MD5 MUST only be used "within" a TLS server-authenticated "pipe", and MUST only be used for client authentication. That is, we do not use the DIGEST-MD5 security services (confidentiality, integrity etc.).

2.3.2. Handling Multiple Credentials for an Account

When more than one credential is stored under a single account, the client can select a single credential using the optional credential selector string.

There is no concept of a "default credential" - all credentials MUST have an associated selector unique for that account. The selector is REQUIRED for upload requests and OPTIONAL for download requests. If the selector is omitted in a download request, it MUST be interpreted as a request for all the stored credentials.

An empty selector string value (i.e. "") in a credential download request is to be interpreted as if the selector string were omitted, i.e. a download request containing this is a request for all credentials.

It is an error to have more than one credential stored under the same account where both have the same credential selector string.

2.3.3. Common Fields

All messages sent to the server MAY contain ProcessInfo values. This field MAY be used by other specifications or for vendor extensions. For example, a server might require clients to include a phone number in this field. The information response message contains a list of the types of ProcessInfo that the server supports. This extensibility scheme is similar to that used in [XKMS] and [XBULK].

Where no specific response message is defined for an operation (e.g. for UploadRequest), then the transport will indicate success or failure.

All of the response messages defined here MAY contain a Status string, containing a value intended for human consumption.

2.3.4. Credential Format

A number of messages involve the Credential element. It has the following fields (all optional fields may occur exactly zero or one times unless otherwise stated):

- CredentialSelector contains a string by which this particular credential (for this account) can be identified.
- PayLoad contains either a ds:KeyInfo or some other form of credential. Implementations MUST support the PKCS #15 form of ds:KeyInfo defined below (the SacredPKCS15 element).
- LastModified is a string containing the time (at the server) at which this credential was last modified.
- TimeToLive (optional) is a hint clients SHOULD honor, which specifies the number of seconds the downloaded credential is to be usable.
- ProcessInfo (optional) MAY contain any (typed) information that the server is intended to process. If the server doesn't support any of the ProcessInfo data, it MAY ignore that data.
- ClientInfo (optional) MAY contain any (typed) information that the client is intended to process, but which the server MUST ignore. If the client doesn't support any of the ClientInfo data, it MAY ignore that data (e.g. if the ClientInfo is device specific).

3. BEEP Profile for SACRED

The protocol described in this memo is realized as a [BEEP] profile.

Future memos may define alternative versions of the BEEP profile for SACRED. When a BEEP peer sends its greeting, it indicates which profiles it is willing to support. Accordingly, when the BEEP client asks to start a channel, it indicates the versions it supports, and if any of these are acceptable to the BEEP server; the latter specifies which profile it is starting.

Profile Identification: http://iana.org/beep/sacred

Messages Exchanged during Channel Creation:

```
     InfoRequest,
     CreateAccountRequest,
     RemoveAccountRequest,
     ModifyAccountRequest,
     DownloadRequest,
     UploadRequest,
     DeleteRequest,
     InfoResponse,
```

DownloadResponse,
error,
ok

Messages starting one-to-one exchanges:
InfoRequest,
CreateAccountRequest,
RemoveAccountRequest,
ModifyAccountRequest,
DownloadRequest,
UploadRequest,
DeleteRequest

Messages in positive replies:
ok,
InfoResponse,
DownloadResponse

Messages in negative replies: error

Messages in one-to-many changes: none

Message Syntax: c.f.,Section 3

Message Semantics: c.f., Section 2

Contact Information: c.f., the editor's address section of this memo

3.1. Profile Initialization

Because all but one of the operations of the SACRED profile have security requirements (cf., Section 2.3.1), before starting the SACRED profile, the BEEP session will likely be tuned using either

http://iana.org/beep/TLS

or

http://iana.org/beep/TLS followed by
http://iana.org/SASL/DIGEST-MD5

Appendix B gives an example of tuning a BEEP session using DIGEST-MD5 (i.e. it shows how to turn on BEEP security).

Regardless, upon completion of the negotiation process, a tuning reset occurs in which both BEEP peers issue a new greeting. Consult Section 3 of [BEEP] for an example of how a BEEP peer may choose to issue different greetings based on whether confidentiality is in use.

Any of the messages listed in section 3.2 below may be exchanged during channel initialization (c.f., Section 2.3.1.2 of [BEEP]), e.g.,

```
C: <start number='1'>
C:   <profile uri='http://iana.org/beep/sacred'>
C:             <![CDATA[<DownloadRequest ...>]]>
C:     </profile>
C: </start>

S: <profile uri='http://iana.org/beep/sacred'>
S:   <![CDATA[<DownloadResponse ...>]]>
S: </profile>
```

Note that BEEP imposes both encoding and length limitations on the messages that are piggybacked during channel initialization.

3.2. Profile Exchange

All messages are exchanged as "application/beep+xml" (c.f., Section 6.4 of [BEEP]):

Role	MSG	RPY	ERR
I	InfoRequest	InfoResponse	error
I	CreateAccountRequest	ok	error
I	RemoveAccountRequest	ok	error
I	ModifyAccountRequest	ok	error
I	DownloadRequest	DownloadResponse	error
I	UploadRequest	ok	error
I	DeleteRequest	Ok	error

3.3. Error Handling

The "error" message from Section 2.3.1.5 of [BEEP] is used to convey error information. Typically, after flagging an error, a peer will initiate a graceful release of the BEEP session.

The following BEEP error reply codes from [BEEP] are to be used:

code	Meaning
421	service not available
450	requested action not taken (e.g., lock already in use)
451	requested action aborted (e.g., local error in processing)

454	temporary authentication failure
500	general syntax error (e.g., poorly-formed XML)
501	syntax error in parameters (e.g., non-valid XML)
504	parameter not implemented
530	authentication required
534	authentication mechanism insufficient (e.g., too weak, sequence exhausted, etc.)
535	authentication failure
537	action not authorized for user
538	authentication mechanism requires encryption
550	requested action not taken (e.g., no requested profiles are acceptable)
553	parameter invalid
554	transaction failed (e.g., policy violation)

The following SACRED-specific error reply codes can also be used:

code	Meaning
555	Extension (ProcessInfo) used not supported
556	Required extension (ProcessInfo) not present
557	StaleCredential (A bad LastModified value was contained in an UploadRequest.)

3.4. SASL Authorization Identity

The use of the SASL authorization identity in this protocol is implementation-specific. If used, the authorization identity is not a substitute for the credential selector field, but may be used to affect authorization for access to credentials.

4. IANA Considerations

The IANA has registered the BEEP profile specified in Section 4.

http://iana.org/beep/sacred

The sacred protocol SHOULD be run over port 1118.

The GSSAPI service name (required when using SASL) for this protocol SHALL be "sacred".

5. Security Considerations

[REQS] calls for specifications to state how they address the vulnerabilities listed below.

V1. A passive attacker can watch all packets on the network and later carry out a dictionary attack.
- The use of DIGEST-MD5 and/or TLS counters this vulnerability.

V2. An attacker can attempt to masquerade as a credential server in an attempt to get a client to reveal information online that allows for a later dictionary attack.
- The use of server or mutual authentication counters this vulnerability.

V3. An attacker can attempt to get a client to decrypt a chosen "ciphertext" and get the client to make use of the resulting plaintext - the attacker may then be able to carry out a dictionary attack (e.g. if the plaintext resulting from "decryption" of a random string is used as a DSA private key).
- The use of server or mutual authentication counters this vulnerability.

V4. An attacker could overwrite a repository entry so that when a user subsequently uses what they think is a good credential, they expose information about their password (and hence the "real" credential).
- Server implementations SHOULD take measures to protect the database. Clients MAY use the ClientInfo field to store e.g. a signature over the Credential, which they then verify before using the private component.

V5. An attacker can copy a credential server’s repository and carry out a dictionary attack.
- Server implementations SHOULD take measures to protect the database.

V6. An attacker can attempt to masquerade as a client in an attempt to get a server to reveal information that allows for a later dictionary attack.
- The mutual authentication requirements of this protocol counter this to a great extent. Additionally, credential servers MAY choose to provide mechanisms that protect against online dictionary attacks against user account passwords, either by repeated access attempts to a single user account (varying the password) or by attempting to access many user accounts using the same password.

V7. An attacker can persuade a server that a successful login has occurred, even if it hasn’t.
- Client authentication prevents this.

V8. (Upload) An attacker can overwrite someone else's credentials on the server.
- Only if they know the account password already (thanks to mutual authentication).

V9. (When using password-based authentication) An attacker can force a password change to a known (or "weak") password.
- Client authentication counters this.

V10. An attacker can attempt a man-in-the-middle attack for lots of reasons...
- Mutual authentication and the encryption of subsequent messages prevents this.

V11. User enters password instead of name.
- Since the DIGEST-MD5 mechanism is only used after TLS tuning, the user's name is also protected.

V12. An attacker could attempt various denial-of-service attacks.
- No specific countermeasures against DoS are proposed.

If the CreateAccountRequest message were sent over a cleartext channel (or otherwise exposed), then an attacker could mount a dictionary attack and recover the account password. This is why the server authenticated TLS transport is REQUIRED for this operation.

If someone steals the server database they can launch a dictionary attack. If the dictionary attack is successful, the attacker can decrypt the user's credentials. An attacker that has learned the user's account password can also upload new credentials, assuming the user is authorized to modify the credentials, because someone who knows the user's account password is assumed to be the user. However, if someone steals the server database and is unsuccessful at obtaining the user's account password through a dictionary attack, they will be unable to upload new credentials.

Credential servers SHOULD incorporate measures that act to counter denial of service attacks. In particular, they SHOULD drop inactive connections and minimize the use of resources by un-authenticated connections. A number of recommendations are listed at [DDOS].

Various operations in the SACRED protocol depend upon server authentication being provided by server authenticated TLS. SACRED clients SHOULD take care that the correct server is at the far end of the TLS "pipe" by performing the checks which are listed in section 3.1 of RFC 2818 [RFC2818]. Clients SHOULD also include the optional BEEP serverName field in their "start" message and SHOULD then ensure that the BEEP serverName is consistent with the checks on the TLS server described in RFC 2818. Failure to carry out these checks could allow a spoof server access to a user's credential.

If the SACRED account password were to be used in some other, less secure protocol, using DIGEST-MD5, then it might appear to be the case that a man-in-the-middle (MITM) attack could be mounted. However, this is not the case since the DIGEST-MD5 client hash includes a client-selected "digest-uri-value", which in SACRED's case will be "sacred/<serverName>". In a MITM attack, those values will be something else. A MITM attack as described is therefore thwarted, because digest-uri-value wouldn't match what the SACRED server is expecting.

6. References

6.1. Normative References

[BEEP] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[DIGEST-MD5] Leach, P. and C. Newman, "Using Digest Authentication as a SASL Mechanism", RFC 2831, May 2000.

[PKCS15] "PKCS #15 v1.1: Cryptographic Token Information Syntax Standard," RSA Laboratories, June 2000.

[REQS] Arsenault, A. and S. Farrell, "Securely Available Credentials - Requirements", RFC 3157, August 2001.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[SASL] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.

[TLS] Dierks, T. and C. Allen, "The TLS Protocol - Version 1.0", RFC 2246, January 1999.

[TLSAES] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[XMLDSIG] Eastlake, 3rd, D., Reagle, J. and D. Solo, "(Extensible Mark-Up Language) XML-Signature Syntax and Processing", RFC 3275, March 2002.

[XMLSCHEMA] "XML Schema Part 1: Structures", D. Beech, M. Maloney, N. Mendelsohn, and H. Thompson. W3C Recommendation, May 2001. Available at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

6.2. Informative References

[DDOS] "Recommendations for the Protection against Distributed Denial-of-Service Attacks in the Internet", http://www.iwar.org.uk/comsec/resources/dos/ddos_en.htm

[RFC2818] Rescorla, E., "HTTP over TLS", RFC 2818, May 2000.

[RFC3760] Gustafson, D., Just, M. and M. Nystrom, "Securely Available Credentials - Credential Server Framework," RFC 3760, April 2004.

[XKMS] Hallam-Baker, P. (ed), "XML Key Management Specification", http://www.w3.org/TR/xkms2/

[XBULK] Hughes, M (ed), "XML Key Management Specification - Bulk Operation", http://www.w3.org/TR/xkms2-xbulk/

Acknowledgements

Radia Perlman (radia.perlman@sun.com) and Charlie Kaufman (charliek@microsoft.com) co-authored earlier versions of this document. Michael Zolotarev (mzolotar@tpg.com.au) did much of the initial work, adapting an earlier version to the use of SRP (though SRP was subsequently dropped, much of the framework survives). Marshall Rose (mrose@dbc.mtview.ca.us) helped out a lot, in particular, with the BEEP profile. And the following people were actively involved in the mailing list discussions leading to this document:

David Chizmadia,
Dave Crocker (dcrocker@brandenburg.com),
Lawrence Greenfield (leg+@andrew.cmu.edu),
Dale Gustafson (degustafson@comcast.net),
Mike Just (just.mike@tbs-sct.gc.ca),
John Linn (jlinn@rsasecurity.com),
Neal McBurnett (neal@bcn.boulder.co.us),
Keith Moore (moore@cs.utk.edu),
RL "Bob" Morgan (rlmorgan@washington.edu),
Magnus Nystrom (magnus@rsasecurity.com),
Eamon O'Tuathail (eamon.otuathail@clipcode.com),
Gareth Richards (grichards@rsasecurity.com)

Of course, any and all errors remain the editor's responsibility.

Appendix A: XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
   <schema
     targetNamespace="urn:sacred-2002-12-19"
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
     xmlns:sacred="urn:sacred-2002-12-19"
     xmlns="http://www.w3.org/2001/XMLSchema">
     <import namespace="http://www.w3.org/2000/09/xmldsig#"
     schemaLocation=
     "http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
     <!-- extensibility holes -->
     <complexType name="ProcessInfoType">
      <sequence maxOccurs="unbounded">
       <any namespace="##other"/>
      </sequence>
     </complexType>
     <element name="ProcessInfo" type="sacred:ProcessInfoType"/>
     <complexType name="ClientInfoType">
      <sequence maxOccurs="unbounded">
       <any namespace="##other"/>
      </sequence>
     </complexType>
     <element name="ClientInfo" type="sacred:ClientInfoType"/>
     <!-- Where to put authenentication information -->
     <complexType name="AuthInfoType">
      <choice maxOccurs="unbounded">
       <element name="DigestMD5AuthInfo">
        <complexType>
         <sequence>
          <element name="PasswordVerifier" type="base64Binary"/>
          <element name="Realm" type="string" />
         </sequence>
        </complexType>
       </element>
       <any namespace="##other"/>
      </choice>
     </complexType>
     <element name="AuthInfo" type="sacred:AuthInfoType"/>
     <!-- authentication mechanism parameters -->
     <complexType name="AuthParamsType">
      <choice maxOccurs="unbounded">
       <element name=" DigestMD5AuthParams">
        <complexType>
         <sequence>
          <element name="Realm" type="string"
            minOccurs="1" maxOccurs="unbounded"/>
         </sequence>
```

```
    </complexType>
   </element>
   <any namespace="##other"/>
  </choice>
 </complexType>
 <element name="AuthParams" type="sacred:AuthParamsType"/>
 <!-- Protocol messsages -->
 <!-- "account handling" operations -->
 <!-- Information request -->
 <element name="InfoRequest"/>
 <element name="InfoResponse">
  <complexType>
   <sequence>
    <element name="Status" type="string" minOccurs="0"/>
    <element name="ServerId" type="string"/>
    <element ref="sacred:AuthParams"/>
    <element ref="sacred:ProcessInfo" minOccurs="0"/>
   </sequence>
  </complexType>
 </element>
 <!-- Create Account Request -->
 <element name="CreateAccountRequest">
  <complexType>
   <sequence>
    <element name="UserId" type="string"/>
    <element ref="sacred:AuthInfo"/>
    <element ref="sacred:ProcessInfo" minOccurs="0"/>
   </sequence>
  </complexType>
 </element>
 <!-- remove account request -->
 <element name="RemoveAccountRequest">
  <complexType>
   <sequence>
    <element ref="sacred:ProcessInfo" minOccurs="0"/>
   </sequence>
  </complexType>
 </element>
 <!-- password change request -->
 <element name="ModifyAccountRequest">
  <complexType>
   <sequence>
    <element ref="sacred:AuthInfo"/>
    <element ref="sacred:ProcessInfo" minOccurs="0"/>
   </sequence>
  </complexType>
 </element>
 <!-- "run-time" operations -->
```

```
<!-- DownLoad Request -->
<element name="DownloadRequest">
 <complexType>
  <sequence>
   <element name="CredentialSelector" type="string"
     minOccurs="0"/>
   <element ref="sacred:ProcessInfo" minOccurs="0"/>
  </sequence>
 </complexType>
</element>
<!-- Download Response -->
<element name="DownloadResponse">
 <complexType>
  <sequence>
   <element name="Status" type="string" minOccurs="0"/>
   <element name="Credential" type="sacred:CredentialType"
    maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
<!-- Upload request -->
<element name="UploadRequest">
 <complexType>
  <sequence>
   <element name="Credential" type="sacred:CredentialType"/>
  </sequence>
 </complexType>
</element>
<element name="DeleteRequest">
  <complexType>
    <sequence>
      <choice>
        <sequence>
          <element name="CredentialSelector" type="string"/>
          <element name="LastModified" type="dateTime"
                minOccurs="0"/>
        </sequence>
        <element name="All"/>
      </choice>
      <element ref="sacred:ProcessInfo" minOccurs="0"/>
    </sequence>
  </complexType>
</element>
<!-- Credential related structures -->
<!-- A new ds:KeyInfo thing -->
<element name="SacredPKCS15" type="base64Binary"/>
<!-- credential -->
<complexType name="CredentialType">
```

```
         <sequence>
          <element name="CredentialSelector" type="string"/>
          <element name="LastModified" type="dateTime"/>
          <element name="Payload" type="ds:KeyInfoType" minOccurs="0"/>
          <element name="TimeToLive" type="string" minOccurs="0"/>
          <element ref="sacred:ProcessInfo" minOccurs="0"/>
          <element ref="sacred:ClientInfo" minOccurs="0"/>
         </sequence>
        </complexType>

   </schema>
```

Appendix B: An Example of Tuning with BEEP

Here is what tuning BEEP for authentication and confidentiality looks like using TLS and SASL's DIGEST-MD5:

```
L: <wait for incoming connection>
I: <open connection>

 ... each peer sends a greeting indicating the services that
    it offers ...

L: RPY 0 0 . 0 233
L: Content-Type: application/beep+xml
L:
L: <greeting>
L:    <profile uri='http://iana.org/beep/SASL/DIGEST-MD5' />
L:    <profile uri='http://iana.org/beep/TLS' />
L:    <profile uri='http://iana.org/beep/sacred' />
L: </greeting>
L: END
I: RPY 0 0 . 0 52
I: Content-Type: application/beep+xml
I:
I: <greeting />
I: END


 ... the initiator starts a channel for TLS and piggybacks a request
    to start the TLS negotiation ...

I: MSG 0 1 . 52 149
I: Content-Type: application/beep+xml
I:
I: <start number='1' serverName="sacred.example.org">
I:    <profile uri='http://iana.org/beep/TLS'>
I:        &lt;ready />
```

```
I:    </profile>
I: </start>
I: END
```

```
 ... the listener creates the channel and piggybacks its readiness to
    start TLS ...
```

```
L: RPY 0 1 . 233 112
L: Content-Type: application/beep+xml
L:
L: <profile uri='http://iana.org/beep/TLS'>
L:     &lt;proceed />
L: </profile>
L: END
```

```
 ... upon receiving the reply, the initiator starts up TLS ...

 ... successful transport security negotiation ...

 ... a new greeting is sent (cf., Section 9 of RFC 3080), note that
    the listener no longer advertises TLS (we're already running
    it)
```

```
L: RPY 0 0 . 0 186
L: Content-Type: application/beep+xml
L:
L: <greeting>
L:    <profile uri='http://iana.org/beep/SASL/DIGEST-MD5' />
L:    <profile uri='http://iana.org/beep/sacred' />
L: </greeting>
L: END
I: RPY 0 0 . 0 52
I: Content-Type: application/beep+xml
I:
I: <greeting />
I: END
```

```
 ... the initiator starts a channel for DIGEST-MD5 and piggybacks
    initialization information for the mechanism ...
```

```
I: MSG 0 1 . 52 178
I: Content-Type: application/beep+xml
I:
I: <start number='1'>
I:    <profile uri='http://iana.org/beep/SASL/DIGEST-MD5'>
I:        &lt;blob> ... &lt;/blob>
I:    </profile>
```

```
I: </start>
I: END

 ... the listener creates the channel and piggybacks a challenge ...

L: RPY 0 1 . 186 137
L: Content-Type: application/beep+xml
L:
L: <profile uri='http://iana.org/beep/SASL/DIGEST-MD5'>
L:     &lt;blob> ... &lt;/blob>
L: </profile>
L: END

 ... the initiator sends a response to the challenge ...

I: MSG 1 0 . 0 58
I: Content-Type: application/beep+xml
I:
I: <blob> ... </blob>
I: END

 ... the listener accepts the challenge and tells the initiator
    that it is now authenticated ...

L: RPY 1 0 . 0 66
L: Content-Type: application/beep+xml
L:
L: <blob status='complete' />
L: END


 ... the initiator starts a channel for SACRED and piggybacks its
    initial SACRED request ...

I: MSG 0 2 . 230 520
I: Content-Type: application/beep+xml
I:
I: <start number='3'>
I:    <profile uri='http://iana.org/beep/sacred' />
I:        &lt;?xml version="1.0" encoding="UTF-8"?>
I:        &lt;sacred:DownloadRequest
I:          xmlns:sacred="urn:sacred-2002-12-19"
I:          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
I:          xsi:schemaLocation="urn:sacred-2002-12-19 sacred.xsd">
I:          &lt;CredentialSelector>
I:                      magnus-credentials&lt;/CredentialSelector>
I:        &lt;/sacred:DownloadRequest>
I: </start>
```

```
I: END

 ... the listener creates the channel and piggybacks the response to
the initial SACRED request

L: RPY 0 2 . 323 805
L: Content-Type: application/beep+xml
L:
L: <profile uri='http://iana.org/beep/sacred' />
L:     &lt;?xml version="1.0" encoding="UTF-8"?>
L:     &lt;sacred:DownloadResponse
L:       xmlns:sacred="urn:sacred-2002-12-19"
L:       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
L:       xsi:schemaLocation="urn:sacred-2002-12-19 sacred.xsd">
L:       &lt;Status>Success&lt;/Status>
L:       &lt;Credential>
L:         &lt;CredentialSelector>
L:              magnus-credential&lt;/CredentialSelector>
L:         &lt;LastModified>2002-11-22T00:00:08Z&lt;/LastModified>
L:         &lt;Payload>
L:             &lt;sacred:SacredPKCS15
L:               xmlns:sacred="urn:sacred-2002-12-19">GpM7
L:             &lt;/sacred:SacredPKCS15>
L:         &lt;/Payload>
L:       &lt;/Credential>
L:     &lt;/sacred:DownloadResponse>
L: </profile>
L: END
```

Appendix C: Provision SACRED using other Protocols

SACRED may be implemented in a non-BEEP environment, provided that before any SACRED PDUs are sent, the application protocol must be protected according to the security mandates provided in Section 2.3.

For example, if SACRED is provisioned as the payload of an application protocol that supports SASL and TLS, then the appropriate SASL and/or TLS negotiation must successfully occur before exchanging Sacred PDUs.

Alternatively, if the application protocol doesn't support SASL, then one or more PDUs are defined to facilitate a SASL negotiation, and the appropriate negotiation must occur before exchanging Sacred PDUs.

Editor's Address

Stephen Farrell,
Distributed Systems Group,
Computer Science Department,
Trinity College Dublin,
IRELAND
Phone: +353-1-608-3070
EMail: stephen.farrell@cs.tcd.ie

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.