Network Working Group
Request for Comments: 3760
Category: Informational

D. Gustafson
Future Foundation
M. Just
Treasury Board of Canada
M. Nystrom
RSA Security

April 2004

Securely Available Credentials (SACRED) - Credential Server Framework

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

As the number, and more particularly the number of different types, of devices connecting to the Internet increases, credential mobility becomes an issue for IETF standardization. This document responds to the requirements on protocols for secure exchange of credentials listed in RFC 3157, by presenting an abstract protocol framework.

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 proposes a general framework for secure exchange of such credentials and provides a high level outline that will help guide the development of one or more securely available credentials (SACRED) credential exchange protocols.

2. Functional Overview

Requirements for SACRED are fully described in [RFC3157]. These requirements assume that two distinctly different network architectures will be created to support credential exchange for roaming users:

a) Client/Server Credential Exchange
b) Peer-to-Peer Credential Exchange

This document describes the framework for one or more client/server credential exchange protocols.

In all cases, adequate user authentication methods will be used to ensure credentials are not divulged to unauthorized parties. As well, adequate server authentication methods will be used to ensure that each client’s authentication information (see Section 2.1) is not compromised, and to ensure that roaming users interact with intended/authorized credential servers.

2.1. Definitions

This section provides definitions for several terms or phrases used throughout this document.

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

client authentication information: information that is presented by the client to a server to authenticate the client. This may include a password token, a registration string that may have been received out-of-band (and possibly used for initially registering a roaming user) or data signed with a signature key belonging to the client (e.g., as part of TLS [RFC2246] client authentication).

credentials: cryptographic objects and related data used to support secure communications over the Internet. Credentials may consist of public/private key pairs, symmetric keys, X.509 public key certificates, attribute certificates, and/or application data. Several standardized formats for the representation of credentials exist, e.g., [PKCS12], [PKCS15] (see "secured credentials" below).

passkey: a symmetric key, derived from a password.

password: a string of characters known only to a client and used for the purposes of authenticating to a server and/or securing credentials. A user may be required to remember more than one password.

password token: a value derived from a password using a one-way function that may be used by a client to authenticate to a server. A password token may be derived from a password using a one-way hash function, for example.

secured credentials: a set of one or more credentials that have been cryptographically secured, e.g., encrypted/MACed with a passkey. Secured credentials may be protected using more than one layer of encryption, e.g., the credential is secured with a passkey corresponding to a user’s password and also by a key known only to the server (the credential’s stored form). During network transfer, the passkey-protected credential may be protected with an additional encryption layer using a symmetric key chosen by the Credential Server (e.g., the transmitted form).

strong password protocol: a protocol that authenticates clients to servers securely (see e.g., [SPEKE] for a more detailed definition of this), where the client need only memorize a small secret (a password) and carries no other secret information, and where the server carries a verifier (password token) which allows it to authenticate the client. A shared secret is negotiated between client and server and is used to protect data subsequently exchanged.

Note the distinction between an "account password" and a "credential password." An account password (and corresponding password token) is used to authenticate to a Credential Server and to negotiate a key that provides session level encryption between client and server.

A credential password is used to derive a passkey that's used to provide persistent encryption and authentication for a stored credential. Applicable secured credential standards documents (e.g., [PKCS15]) describe the technical details of specific password-based-encryption (pbe) techniques that are used to protect credentials from unauthorized use.

Although the same password value may be used to provide both services, it is likely that different, algorithm specific passkeys would be generated from this password (i.e., because of different salt values, etc.).

In addition, although it may be more convenient for a user to remember only a single password, differing security policies (e.g., password rules) between the credential server and the credential issuers may result in a user having to remember multiple passwords.

2.2. Credentials

This document is concerned with the secure exchange and online management of credentials in a roaming or mobile environment. Credentials MAY be usable with any end user device that can connect to the Internet, such as:

- desktop or laptop PC
- mobile phone
- personal digital assistant (PDA)
- etc.

The end user system may, optionally, store its credential information on special hardware devices that provide enhanced portability and protection for user credentials.

Since the credential usually contains sensitive information that is known only to the credential holder, credentials MUST NOT be sent in the clear during network transmission and SHOULD NOT be in the clear when stored on an end user device such as a diskette or hard drive. For this reason, a secured credential is defined. Throughout this document we assume that, at least from the point of view of the protocol, a secured credential is an opaque (and at least partially privacy and integrity protected) data object that can be used by a network connected device. Once downloaded, clients must be able to recover their credentials from this opaque format.

At a minimum, all supported credential formats SHOULD provide privacy and integrity protection for private keys, secret keys, and any other data objects that must be protected from disclosure or modification. Typically, these security capabilities are part of the basic credential format such that the credential (e.g., a data file) is protected when stored on hard drives, flexible diskettes, etc.

During network transmission, the secured credential is protected with a second (outer) encryption layer. The outer encryption layer is created using a session-level encryption key that was derived during the mutual authentication process. Effectively, secured credentials traverse an "encrypted tunnel" that provides an additional layer of privacy protection for credentials (and any other) information exchanged.

2.3. Network Architecture

The network diagram below shows the components involved in the SACRED client/server framework.

```
                     +--------+           +------------+
                     | Client +-----------| Credential |
                     +--------+     1     |   Server   |
                          \               +-----+------+
                           \                    |
                            \                   | 2
                             \                  |
                              \    3      +-----+------+
                               -----------| Credential |
                                          |  Store(s)  |
                                          +------------+
```

Client - The entity that wants to retrieve their credentials from a credential server.

Credential Server - The server that downloads secure credentials to and uploads them from the client. The server is responsible for authenticating the client to ensure that the secured credentials are exchanged only with an appropriate end user. The credential server is authenticated to the client to ensure that the client’s authentication information is not compromised and so that the user can trust the credentials retrieved.

Credential Store - The repository for secured credentials. There might be access control features but those generally aren't sufficient in themselves for securing credentials. The credential server may be capable of splitting credentials across multiple credential stores for redundancy or to provide additional levels of protection for user credentials.

Protocol 1 - The protocol used to authenticate the client and credential server, and download and upload user credentials from a credential server.

Protocol 2 - The protocol used by the Credential Server to store and retrieve user credentials (LDAP, LDAP/SSL, or other).

Protocol 3 - The protocol used by the client to store and retrieve user credentials from the credential store (LDAP, LDAP/SSL, or other).

This framework describes the high level design for protocol 1. Protocols 2 and 3 are closely related (but out of scope for this document) and could be implemented using standard protocols, such as LDAP or secure LDAP, or other standard or proprietary protocols. Note also that any administrator-credential server protocols are assumed to be server vendor specific and are not the subject of SACRED standardization efforts at this time.

Clients are not precluded from exchanging credentials directly with a credential store (or any other server of it's choosing). However, mutual authentication with roaming users and a consistent level of protection for credential data while stored on network servers and while in transit is provided by SACRED protocols exchanged with the credential server. Depending on credential server design, user credentials may flow through the credential server to the credential store or directly between the client and the credential store.

Also, users may upload their credentials to several credential servers to obtain enhanced levels of availability. Coordination (automatic replication) of user information or credential data among several credential servers is currently beyond the scope of this document.

3. Protocol Framework

This section provides a high level description of client/server protocols that can be used to exchange and manage SACRED credentials.

The client/server credential exchange protocol is based on three basic and abstract operations; "GET", "PUT", and "DELETE". The secured credential exchange protocol is accomplished as follows:

connect - the client initiates a connection to a credential server for the purpose of secure credential exchange.

mutual authentication/key negotiation - using a strong password protocol (or equivalent) the client authenticates to the server, the server authenticates to the client, and a session level encryption key is negotiated. The details of the mutual authentication protocol exchange are dependent upon the particular authentication method used. In all cases, the end result is to authenticate the client to the server and server to the client, and establish a strong, shared secret between the two parties.

client request(s) - the SACRED client issues one or more high level credential exchange requests (e.g., GET, PUT, or DELETE).

server response(s) - the SACRED credential server responds to each request, either performing the operation successfully or indicating an appropriate error.

close - the client indicates it has no more requests for the server at this time. The security context between client and server is no longer needed. Close is a logical, session management operation.

disconnect - the parties disconnect the transport level connection between client and server. Note that "connect" and "disconnect" are logical, transport-layer dependent operations that enclose the protocol exchange between the two communicating processes.

Each high-level credential exchange operation is made up of a series of request-response pairs. The client initiates each request, which the server processes before returning an appropriate response. Each request must complete (server reports success or failure) before the client issues the next request. The server SHOULD be willing to service at least one upload or download request following successful mutual authentication but either party can terminate the logical connection at any time.

In the following sections, secured credentials and related values are represented using the following notation:

SC-x is the secured credential file, which includes a format identifier field and credential data. The credential data is an opaque, encrypted data object (e.g., PKCS#15 or PKCS#12 file). The format identifier is needed to correctly parse the credential data.

Name-x is an account-defined selector or locator (a user friendly name) that is used to indicate a specific secured credential. The name of each credential stored under a given user account MUST be unique e.g., there may be one credential called "financial" and another called "healthcare", etc. At a minimum, credential names MUST be unique across a given account/user name. When no name is supplied for a GET operation, all credentials stored for the given username will be returned.

ID-x is a distinct credential version indicator that MAY be used to request a conditional GET/PUT/DELETE operation. This credential-ID value SHOULD contain the server’s "last-modified" date and time (e.g., the time that this particular credential version was stored on the server) and MAY contain additional information such as a sequence number or a (complete or partial) credential fingerprint that is used to ensure the credential-ID is unique from other credential versions stored under the same user account and credential name.

All named credentials may be accessed by authenticating under a single username. If a user needs or prefers to use more than one distinct authentication password (and/or authentication method) to protect access to several secured credentials, he/she SHOULD register those credentials under distinct user/account names, one for each different authentication method used.

3.1. Credential Upload

The purpose of a credential upload operation is to allow a client to register new credentials, or replace currently stored credentials (e.g., credentials that may have been updated by the client using appropriate key management software).

The framework for the credential upload, as implemented using the PUT operation, is:

- The client and server establish a mutually authenticated session and negotiate a shared secret.

- The client will then issue a PUT message that contains the upload credential and related data fields.

- The server will respond to the PUT, indicating the credential was successfully stored on the server or that an error occurred.

The client's PUT request MAY contain an optional identifier (credential-ID) field. If present, the new credential will only be stored if a credential with the same name and credential-ID is currently stored on the server (e.g., a logical REPLACE operation is performed). The server MUST return an error if a client attempts to replace a credential that does not exist on the server.

The credential server's response to a PUT request MUST contain a credential version identifier (credential-ID) for the newly stored credential that MAY be used by clients to optimize subsequent download operations and avoid credential version mismatches.

3.1.1. Credential Upload Protocol Sequence

The following gives an example of a "credential upload" protocol sequence:

```
     client                               server
     -------                              -------

     < connect >                  -->

     <--- mutual authentication --->

     < PUT SC-1, Name-1, [ID-1] > -->
                                  <--     < Name-1, new-ID-1 >
     < PUT SC-2, Name-2, [ID-2] > -->
                                  <--     < Name-2, new-ID-2 >

                                  ...

     < close >                    -->
                                  <--     OK (+ disconnect)
```

new-ID-x is the credential-ID of the newly stored credential.

3.2. Credential Download

Roaming clients can download their credentials at any time after they have been uploaded to the server.

The framework for a credential download, as implemented using the GET operation, is:

- The client SHOULD authenticate the server.

- The user MUST be authenticated (by the server).

- A GET request for the credential download is issued.

- The response contains the credential and format identifier.

The specific user credential being requested may be identified by name in the message sent to the credential server. If successful, the response MUST contain the requested credential data element (format ID and data) as defined above.

If the user issues a GET request with a NULL credential name field, the server SHOULD return all credentials stored under the current user account.

Optionally, the client MAY include a credential-ID to indicate a conditional download request. In this case, the server will return the requested credential if and only if the ID of the credential currently stored on the server does NOT match the ID specified.

The server should return either the requested credential or a distinct response indicating that the conditional download was not performed (e.g., the client already has a copy of this exact credential).

3.2.1. Credential Download Protocol Sequence

The following gives an example of a "credential download" protocol sequence:

```
          client                        server
          -------                      --------

        < connect >             -->

        <--- mutual authentication -->

        < GET Name-1, [ID-1] >  -->
                               <--     < SC-1, ID-1’ >
        < GET Name-2, [ID-2] >  -->
                               <--     < GET response >

                               ...

        < close >              -->
                               <--     OK (+ disconnect)
```

Notice that for the second request, no credential has been returned since ID-2, as included in the client’s request, matched the identifier for the Name-2 credential.

3.3. Credential Removal

The framework for the credential removal, as implemented with the DELETE operation, is:

- The credential server MUST be authenticated (by the client) using a method-dependent protocol sequence.

- The user MUST be authenticated (by the server) using a method-dependent protocol sequence.

- The user then sends a DELETE request message that contains the credential name indicating which credential to remove.

- Optionally, the client may include a credential-ID in the DELETE request. In this case, the credential will be deleted if the request ID matches the ID of the credential currently stored on the server. This may be done to ensure that a client intending to delete their stored credential does not mistakenly delete a different version of the credential.

3.3.1. Credential Removal Protocol Sequence

The following gives an example of a "credential removal" protocol sequence:

```
      client                             server
      -------                           --------

    < connect >                --> 

    <-------- mutual authentication -------->

    < DEL Name-1, [ID1] >      --> 
                               <--     < Name-1 deleted >
    < DEL Name-2, [ID2] >      --> 
                               <--     < Name-2 deleted >

                               ...

    < close >                  --> 
                               <--     OK (+ disconnect)
```

3.4. Credential Management

Note that the three operations defined above (GET, PUT, DELETE) can be used to perform the basic credential management operations:

- add a new credential on the server,
- update (replace) an existing credential, and
- delete an existing credential.

The information provided for these basic operations might be used to help guide the design of more complex operations such as user registration (add account), user deregistration (remove account), change account password, or list all credentials.

Note that, in the case where a credential with the same name exists on the server, uploading a NULL credential is logically equivalent to removing a previously stored credential.

4. Protocol Considerations

4.1. Secure Credential Formats

To ensure that credentials created on, and uploaded from, one device can be downloaded and used on any other device, there is a need to define a single "mandatory to implement" credential format that must be supported by all conforming client implementations.

At least two well-defined credential formats are available today: [PKCS12] and [PKCS15].

Other optional credential formats may also be supported if necessary. For example, additional credential formats might be defined for use with specific (compatible) client devices. Each credential format MUST provide adequate privacy protection for user credentials when they are stored on flexible diskettes, hard disks, etc.

Throughout this document, the credential is treated as an opaque (encrypted) data object and, as such, the credential format does not affect the basic credential exchange protocol.

4.2. Authentication Methods

Authentication is vitally important to ensure that credentials are accepted from and delivered to the authorized end user only. If an unsecured credential is delivered to some other party, the credential may be more easily compromised. If a credential is accepted from an unauthorized party, the user might be tricked into using a credential that has been substituted by an attacker (e.g., an attacker might replace a newer credential with an older credential belonging to the same user).

Ideally, the list of authentication methods should be open ended, allowing new methods to be added as needs are identified and as they become available. For all credentials, the user authentication method and data is defined when a user is first registered with the credential server and may be updated from time to time thereafter by the authorized user.

To adequately protect user credentials from unauthorized disclosure or modification in a roaming environment, all SACRED authentication methods MUST provide protection for user credentials in network environments where attackers might attempt to exploit potential security vulnerabilities. See SACRED Requirements [RFC3157], Section 3.1, Vulnerabilities.

At a minimum, each SACRED authentication method SHOULD ensure that:

- The server authenticates the client
- The client authenticates the server
- The client and server securely negotiate (or derive) a cryptographically strong, secret key (e.g., a session key).
- The exchange of one or more user credentials is protected using this session key.

It is expected that all SACRED client/server protocols will provide each of these basic security functions. Some existing authentication protocols that might be used for this purpose include:

- Strong password protocols
- TLS

Sections 4.2.1 and 4.2.2 provide some guidance about when to use these authentication methods based on the generic security capabilities they provide and the security elements (passwords, key pairs, user certificates, CA certificates) that must be available to the SACRED client.

4.2.1. Strong Password Protocols

Strong password protocols such as those described in [RFC2945], [BM92], [BM94], and [SPEKE] MAY be used to provide mutual authentication and privacy for SACRED protocols.

All strong password protocols require that user-specific values (i.e., a passtoken and related values) be configured within the server. Only a party who knows the password can calculate the verifier value. It must be securely delivered to the server at a time when the client establishes a relationship with the server. At connect time, messages are exchanged between the two parties and complementary algorithms are used to compute a shared common value known only to the legitimate user and the server. Both parties derive a strong (symmetric) key that may be used to secure communications between the two parties.

4.2.2. TLS Authentication

TLS authentication may either be mutual between the client and server or unilateral where only the server is authenticated to the client. These options are described in the next two subsections.

In both cases, TLS can be used to authenticate the server whenever the TLS client has been pre-configured with the necessary certificates needed to validate the server’s certificate chain (including revocation status checking).

TLS Server Authentication (sTLS)

TLS provides a basic secure session capability (sometimes called server-side TLS) whereby the client authenticates the server and a pair of session level encryption keys is securely exchanged between client and server. Following server authentication and security context setup, all client requests and server responses exchanged are integrity and privacy protected.

Protocol designers and implementors should be aware that the flexibility of the certificate-based TLS server authentication method creates security risks that need to be mitigated. Specifically, the need to ensure the user is connected to the intended credential server (secure site), and no other. The TLS v1.0 standard [RFC2246] identifies the basis for managing this risk in section F.3 (see also Section 5.2 in this document):

> "Implementations and users must be careful when deciding which certificates and certificate authorities are acceptable; a dishonest certificate authority can do tremendous damage."

Note also that a faulty implementation of (increasingly complex) TLS server certificate chain processing, by the SACRED client, could lead to similar compromise, allowing successful credential server masquerade or man-in-the-middle attacks.

An engineering approach that provides an enhanced or augmented server authentication method may be warranted for SACRED protocol designs. It is also important to understand that simple layering of independently developed security protocols (e.g., using BEEP or similar layering techniques) produces a complex, multilayer security protocol that might be easily defeated by a combination-specific attack that is able to expose and exploit known weaknesses of the individual protocol(s).

When necessary, and after a TLS session has been established between the two parties, the credential server can request that the client provide her user id and password information to authenticate the remote user. Preferably, client and server can cooperate to perform an authentication operation that allows the server to authenticate the client (and perhaps vice-versa) in a "zero knowledge manner". In such cases, the client need not have a security credential.

TLS with Client Authentication (cTLS)

TLS provides an optional, secure session capability (sometimes called client-side TLS) whereby the TLS server can request client authentication by verifying the client’s digital signature.

In order to use cTLS to provide mutual authentication, the client must also be configured with at least one security credential that is acceptable to the TLS server for remote client authentication purposes.

4.2.3. Other Authentication Methods

Other authentication methods that provide the necessary security capabilities MAY also be suitable for use with SACRED credential exchange protocols.

4.3. Transport Protocol Suites

It is intended that one or more underlying protocol stacks may carry the SACRED credential exchange protocols. It is recognized at the outset that the use of several underlying protocol suites, although not ideal from an interoperability standpoint, may well be required to support the wide variety of needs anticipated.

The SACRED list members have discussed several protocol suites that have been considered on their technical merits, each with distinct benefits and protocol design/implementation costs. Among these protocols are:

- TCP
- BEEP
- HTTP

All protocol suites listed here depend on TCP to provide a reliable, end-to-end transport layer protocol. Each of these building block approaches provides a different way of handling the remaining application layer issues (basic session management, session level security, presentation/formatting, application functionality).

4.3.1. TCP

This approach (layering a SACRED credential exchange protocol directly on top of a TCP connection) requires the development of a custom credential exchange messaging protocol that interfaces to a TCP connection/socket. The primary benefit of this approach is the ability to provide exactly the protocol functionality needed and no more. Most server and client development environments already provide the socket level API needed.

4.3.2. BEEP

This approach builds on the Blocks Extensible Exchange Protocol (BEEP) described in [RFC3080]. BEEP provides general purpose, peer-to-peer message exchange over any of several transport mechanisms where the necessary transport layer mappings have been defined for operation over TCP, TLS, etc. See also [RFC3081].

BEEP provides the necessary user authentication/session security and session management capabilities needed to support SACRED credential exchange operations.

4.3.3. HTTP

This approach builds on the Hypertext Transport Protocol (HTTP) described in [RFC1945] and [RFC2616]. HTTP provides general purpose typing and negotiation of data representation, allowing systems to be built independently of the data objects being transferred. HTTP support is available in a wide variety of server and client platforms, including portable devices that apply to roaming environments (laptop PCs, PDAs, mobile phones, etc.).

HTTP is layered over TCP and can be used, optionally, with TLS to provide authenticated, session level security. Either or both TLS authentication options, sTLS or cTLS, may be used whenever TLS is supported.

5. Security Considerations

The following security considerations identify general observations and precautions to be considered for a framework supporting credential mobility. When designing or implementing a protocol to support this framework, one should recognize these security considerations, and furthermore consult the SACRED Requirements document [RFC3157] Security Considerations.

5.1. Communications Security

A SACRED PDU will contain information pertaining to client or server authentication, or communication of credentials. This information is subject to the traditional security concerns identified below.

5.1.1. Confidentiality

The password or password verifier should be protected when communicated from the client to credential server. The communicated value should be resistant to a dictionary attack.

Similarly, the entity credentials must be confidentiality protected, when communicated from the client to the server and vice-versa. The communicated value should also resist a dictionary attack.

5.1.2. Integrity

Communication integrity between the client and the credential server is required. In this way, intended client operations may not be altered (e.g., from an update to a deletion of credentials), nor may clients be maliciously given "old" credentials (e.g., possibly by an attacker replaying a previous credential download).

5.1.3. Entity Authentication

Proper authentication of the client and server is required to achieve communication confidentiality and integrity.

The server must properly authenticate the client, so that credentials are not mistakenly revealed to an attacker. The client must ensure the proper identification of the credential server so as to prevent revealing their password to an attacker. These goals may be achieved implicitly with a strong password-based protocol or explicitly. If the server is identified explicitly, the user or client must ensure that the user password is conveyed to a trusted server. This might be achieved by installing appropriate trusted key(s) in the client.

5.1.4. Non-repudiation

There are no requirements upon the SACRED protocol itself to support non-repudiation, although the context in which the credentials are being used may have such requirements.

5.2. Systems Security

Systems security is concerned with protection of the protocol endpoints (i.e., the client and server) and information stored at the server in support of the SACRED protocol.

5.2.1. Client Security

As with most security protocols, secure use of the client often relies, in part, upon secure behavior by the user. In the case of a password-based SACRED protocol, users should be educated, or enforced through policy, to choose passwords with a reasonable amount of entropy. Additionally, users should be made aware of the importance of protecting the confidentiality of their account password.

In addition, the client interface should be designed to thwart "shoulder surfing" where an attacker can observe the password as entered by a user. This is often achieved by not echoing the exact characters of the password when entered.

As well, the interface should encourage the entering of the password in the appropriate interface field so that protections can be properly enforced. For example, a user should be guided to not mistakenly enter their password in the "username" field (since their password would likely be echoed to the screen in this case, and might not be encrypted when communicated to the server). This might be accomplished via the automatic insertion of the user name or several user name choices in the appropriate on-screen dialog field, for example.

5.2.2. Client Security, TLS Server Authentication

When TLS is used as the SACRED transport protocol, the client interface should be designed to allow the user to verify that she is connected to the intended credential server. For example, client software should allow for the visual display of identifying components from the TLS server’s X.509 certificate, like the server’s name, the certificate fingerprint, etc.

Users should be guided to verify this information regularly, allowing ready recognition of trusted credential servers. In addition, users should be made aware of the importance of verifying their credential server’s identity before initiating any credential exchange operations.

A SACRED client SHOULD only be configured with those SACRED trust anchors that are to be used by the client. Re-use of trust anchors from other applications, e.g., Internet browsers is NOT RECOMMENDED.

5.2.3. Server Security

Password verifiers and user credentials must be afforded a high level of protection at the credential server. In addition to salting and super-encrypting each (to ensure resistance to offline dictionary attacks), a system should ensure that credential server keys are protected using sufficient procedural and physical access controls.

The login to the credential server should be resistant to replay attacks.

Online attempts to access a particular user account should be controlled, or at least monitored. Control might be enforced by incorporating a time delay after a number of unsuccessful logins to a particular account, or possibly the locking of the account altogether. Alternatively, one might simply log unsuccessful attempts where an administrative notice is produced once a threshold of unsuccessful credential access attempts is reached.

5.2.4. Denial of Service

As with most protocols, Denial of Service (DoS) issues must also be considered. In the case of SACRED, most DoS issues are a concern for the underlying transport protocol. However, some concerns may still be mitigated.

Service to a user might be denied in case their account is locked after numerous unsuccessful login attempts. Consideration of protection against online attacks must therefore be considered (as described above). Proper user authentication should ensure that an attacker does not maliciously overwrite a user’s credentials. Credential servers should be wary of repeated logins to a particular account (which also identifies a possible security breach, as described above) or abnormal volumes of requests to a number of accounts (possibly identifying a DoS attack).

6. References

6.1. Normative References

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

[RFC3157] Arsenault, A. and S. Farrell, "Securely Available Credentials - Requirements", RFC 3157, August 2001.

6.2. Informative References

[BM92] Bellovin, S. and M. Merritt, "Encrypted Key Exchange: Password-based protocols secure against dictionary attacks", Proceedings of the IEEE Symposium on Research in Security and Privacy, May 1992.

[BM94] Bellovin, S. and M. Merritt, "Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise, ATT Labs Technical Report, 1994.

[PKCS12] "PKCS 12 v1.0: Personal Information Exchange Syntax", RSA Laboratories, June 24, 1999.

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

[RFC1945] Berners-Lee, T., Fielding, R. and H. Frystyk, "Hypertext Transfer Protocol-- HTTP/1.0", RFC 1945, May 1996.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frysyk, H., Masinter, L., Leach, M. and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, June 1999.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, September 2000.

[RFC3080] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[RFC3081] Rose, M., "Mapping the BEEP Core onto TCP", RFC 3081, March 2001.

[SPEKE] Jablon, D., "Strong Password-Only Authenticated Key Exchange", September 1996.

7. Authors' Addresses

Dale Gustafson
Future Foundation Inc.

EMail: degustafson@comcast.net

Mike Just
Treasury Board of Canada, Secretariat

EMail: Just.Mike@tbs-sct.gc.ca

Magnus Nystrom
RSA Security Inc.

EMail: magnus@rsasecurity.com

8. Full Copyright Statement

Intellectual Property

Acknowledgement

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