Singularity
Design Note
39

Credentials Manager

Design of the client-side credentials manager service.

Singularity applications and services need to communicate with network services, and to do so they must often be able to authenticate on behalf of a user. The Credentials Manager Service allows applications to perform authentication exchanges. It manages in-memory credentials, implements authentication protocols, and safely exposes access to these authentication protocols to network applications.

The Credentials Manager Service is analogous to some of the functions performed by the Local Security Authority (LSA) service on Windows, and the SSPI API used to access them.

1. Introduction

This document describes the Credentials Manager Service, the contracts used to communicate with it, and the associated command-line tool used to control the service.

2. Requirements

2.1. Sensitive credentials information, such as clear-text passwords, should not be exposed.

Applications should be able to perform authentication exchanges, without having direct access to sensitive information, such as private keys, clear-text passwords (referred to as “evidence”), etc. The Credentials Manager never allows evidence to flow out of the CM process. Instead, authentication protocols run within the CM process, and these protocols exchange authentication tokens with the authenticating process.

2.2. Applications should be isolated from the details of the implementation of authentication protocols.

Application designers should be able to focus on the details of their application protocols, and should not be unnecessarily burdened with the details of authentication protocols or credentials management. For example, the current implementation of the SMB client requires that the user specify a username and password every time the user connects to a share. There is no sense of a login session, as there is on Windows, which allows applications to make use of the user’s credentials.

2.3. Multiple authentication protocols should be supported

Different application protocols can use different authentication protocols. Windows supports NTLM, Kerberos, MD5 CHAP, and more. The Credentials Manager should support multiple authentication protocols, and in a way that allows for easy expansion in the future.

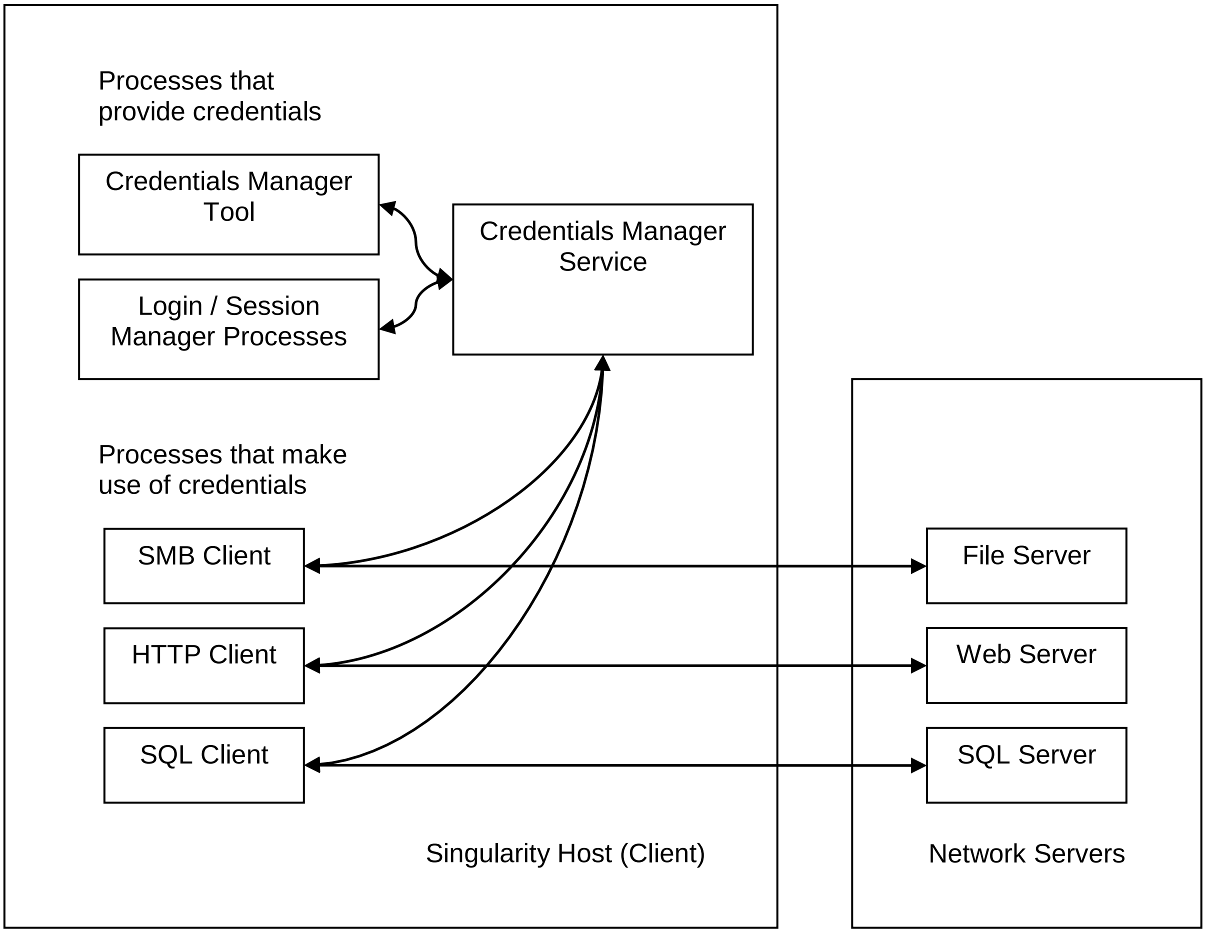
2.4. Users should be able to provide multiple different credentials, and should be able to choose the conditions when those credentials are used.

Users may need to interact with many different network services, and the user may need to use different credentials for specific services (machines) or sets of services. The Credentials Manager should support this.

3. Design

The design consists of these components:

- The Credentials Manager Service. This is a singleton service process. (Once Singularity supports multiple login sessions, it may make sense to create more than one instance of this process. FFS.) (CredentialsManager.exe)
- A command-line tool, used for controlling the CM service. Users can add credentials (provide a username and password), view existing credentials, delete entries, etc. Users can also manage the set of *protocol mappings*, which allow applications to select the correct credentials to use with a particular network service. (cred.exe)
- A library for use by applications, which provides access to the Credentials Manager. This library allows applications (and services) to perform authentication exchanges with network services. (CredentialsManager.Library.dll.)
- The contracts that are used by the CM service. (CredentialsManager.Contracts.dll)


Processes that provide credentials
Credentials Manager Tool
Login / Session Manager Processes
Credentials Manager Service
Processes that make use of credentials
SMB Client
HTTP Client
SQL Client
File Server
Web Server
SQL Server
Singularity Host (Client)
Network Servers

3.1. Contracts

The Credentials Manager uses several different contracts to communicate with clients.

3.1.1. CredentialsManagerContract

This contract is the first point of contact that any process has with the CM service. Clients can connect to the CM using this contract at a well-known path, currently /dev/credentials-manager.

Clients may use this contract to do any of the following:

- Manage the credentials store, which is an in-memory set of credentials managed by the CM. Clients can add credentials to the store (including evidence, such as clear-text passwords, private keys, etc.), delete credentials, or view credentials (excluding evidence).
- Create instances of authentication protocols, using credentials from the credentials store. The instances are known as *supplicants*. This allows applications to perform authentication exchanges with network services. Each authentication protocol defines its own protocol-specific contract or contracts.
- Manage the Protocol Mappings, which is an in-memory set of policies that allow applications to select the correct credentials to use when contacting a network service.

The CredentialsManagerContract makes use of several exchangeable types, which are described below.

3.1.1.1. Exchangeable Type – ProtocolTuple

```
public rep struct ProtocolTuple : ITracked
{
     public char[]! in ExHeap ApplicationProtocol;
     public char[]! in ExHeap ServiceAddress;
     public char[]! in ExHeap AuthenticationProtocol;
     public char[]! in ExHeap Realm;
}
```

The `ProtocolTuple` structure represents primary key of the Protocol Mapping Table.

The `ApplicationProtocol` field identifies the application, such as “smb” or “ftp” or “http”. The Credentials Manager does not impose any constraints on this field, nor does it interpret its contents. Clients of the Credentials Manager service can choose any value for this field.

The `ServiceAddress` field identifies the network address of the remote network service. This field may be any protocol-specific identifier. It may be a DNS domain name, or a NetBIOS name, or an IP address, etc. The Credentials Manager service does not interpret this field. Its structure is determined by each network application that makes use of this service.

The `AuthenticationProtocol` field identifies the authentication protocol that the network application wishes to use when communicating with a remote network service. These strings are defined by the Credentials Manager service. Currently, the only supported protocol is NTLM, identified by the string `ntlm`.

The `Realm` field identifies the authentication realm of the remote network service. Not all authentication protocols use this field; those that do not can specify any value for this field. For NTLM, this is the domain name of the remote network service.

3.1.1.2. Exchangeable Type – CredentialsId

```
public rep struct CredentialsId : ITracked
{
     public char[]! in ExHeap CredentialsName;
     public char[]! in ExHeap Tag;
}
```

This type identifies a set of credentials that are managed by the Credentials Manager Service. It identifies credentials, but does not contain the evidence (such as clear-text passwords, etc.) associated with that identity.

3.1.1.3. AddCredentials Request

```
in message AddCredentials(CredentialsId id, char[]! in ExHeap
password, bool replace);
out message Ok();
out message RequestFailed(CredError error);
```

Adds credentials to the Credentials Store, including the evidence for the credentials. Currently, the only supported form of evidence is clear-text passwords.

If the `replace` parameter is set to true, then the request will replace any existing entry that matches the same credentials ID. If the `replace` parameter is set to false, the service will refuse to replace any existing entry, and the request will complete with an error code.

3.1.1.4. DeleteCredentials Request

```
in message DeleteCredentials(CredentialsId id);
out message Ok();
out message RequestFailed(CredError error);
```

Deletes the identified credentials from the Credentials Store. The credentials name and tag must both match exactly those of the stored credentials.

3.1.1.5. DeleteAllCredentials Request

```
in message DeleteAllCredentials();
out message Ok();
out message RequestFailed(CredError error);
```

Deletes all credentials from the Credentials Store.

3.1.1.6. EnumerateCredentials Request

```
in message EnumerateCredentials();
out message CredentialsList(CredentialsId[]! in ExHeap list);
out message RequestFailed(CredError error);
```

Retrieves a list of the credentials in the Credentials Store. Only the credential identifiers are returned. There is no way to retrieve the evidence for credentials.

3.1.1.7. AddProtocolMapping Request

```
in message AddProtocolMapping(ProtocolTuple tuple, CredentialsId id);
out message Ok();
out message RequestFailed(CredError error);
```

Adds an entry to the Protocol Mapping Table. See the section on the Protocol Mapping Table.

3.1.1.8. DeleteProtocolMapping Request

```
in message DeleteProtocolMapping(ProtocolTuple tuple);
out message Ok();
out message RequestFailed(CredError error);
```

Deletes an entry from the Protocol Mapping Table. The protocol tuple must match exactly an existing entry. Wildcards are never interpreted.

3.1.1.9. DeleteAllProtocolMappings Request

```
in message DeleteAllProtocolMappings();
out message Ok();
out message RequestFailed(CredError error);
```

Deletes all of the entries in the Protocol Mapping Table.

3.1.1.10. EnumerateProtocolMappings Request

```
in message EnumerateProtocolMappings();
out message ProtocolMappings(ProtocolMapping[]! in ExHeap list);
out message RequestFailed(CredError error);
```

Returns a list of all of the entries in the protocol mapping table. There is no defined order for the entries.

3.1.1.11. FindMatchingProtocolMapping Request

```
in message FindMatchingProtocolMapping(ProtocolTuple tuple, bool
useWildcards);
out message NoMatchingProtocolMapping();
out message MatchingProtocolMapping(CredentialsId credentials);
```

This request searches the Protocol Mapping Table for the best credentials to use, using the specified protocol tuple. This allows network applications to choose the appropriate credentials to use, without requiring each network application to re-implement the same credentials selection logic.

3.1.1.12. CreateSupplicant Request

```
in message CreateSupplicant(
     char[]! in ExHeap authenticationProtocol,
     CredentialsId credentials,
     ServiceContract.Exp:Start! exp);
out message RequestFailed(CredError error);
```

This request creates an instance of an authentication protocol, called a supplicant. Network applications can use the supplicant to perform authentication exchanges with remote network services.

The `authenticationProtocol` argument identifies the authentication protocol to use. These values are defined by the Credentials Manager Service. The contracts library provides a class, `AuthenticationProtocolNames`, which contains literal strings that identify the supported authentication protocols; application designers should use references to these literal strings, rather than hard-coding strings into the applications.

The `credentials` argument identifies the credentials to use. The credentials must already have been stored in the Credentials Store.

The `exp` argument provides the export endpoint of a channel. Once the request succeeds, the network application uses that channel to communicate with the supplicant implementation. The contract type used depends on the specific authentication protocol being used. Since the client creates the channel, the client may choose any contract that the authentication protocol implements.

3.1.1.13. CreateSupplicantForProtocol Request

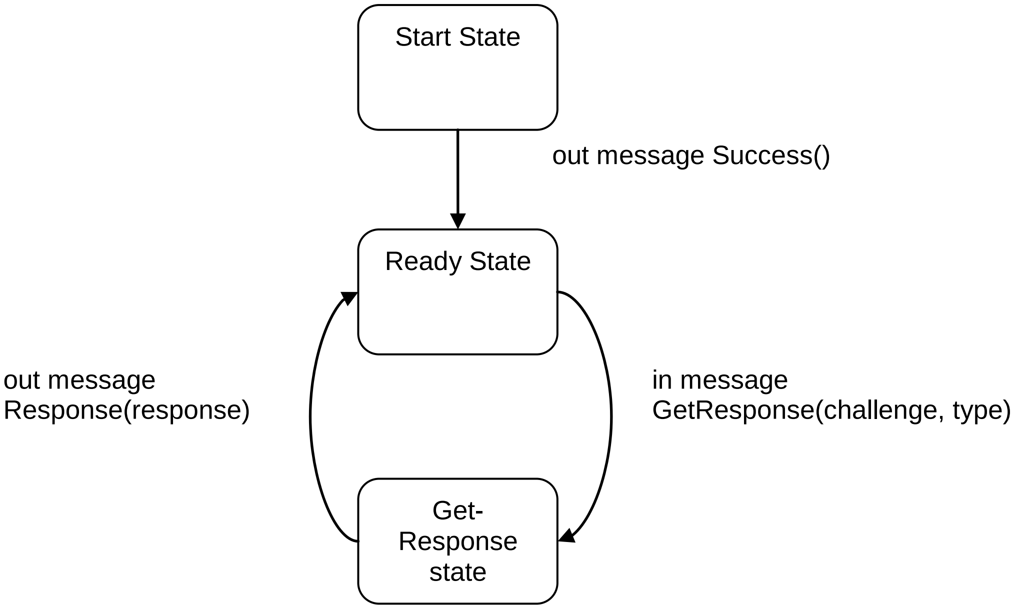
```
in message CreateSupplicantForProtocol(
     ProtocolTuple protocol,
     ServiceContract.Exp:Start! exp);
out message AckCreateSupplicantForProtocol(
     CredentialsId credentialsSelected);
out message RequestFailed(CredError error);
```

This request combines the `FindMatchingProtocolMapping` and `CreateSupplicant` requests into a single request. The client provides a protocol tuple and a channel endpoint. If successful, the Credentials Manager selects credentials, creates a supplicant, and sends a response message, which indicates which credentials were selected.

3.1.2. NtlmSupplicantContract

This contract provides access to the NTLM authentication protocol implementation. The contract provides only a single request, GetResponse, which computes the response to a challenge issued by a remote network service.

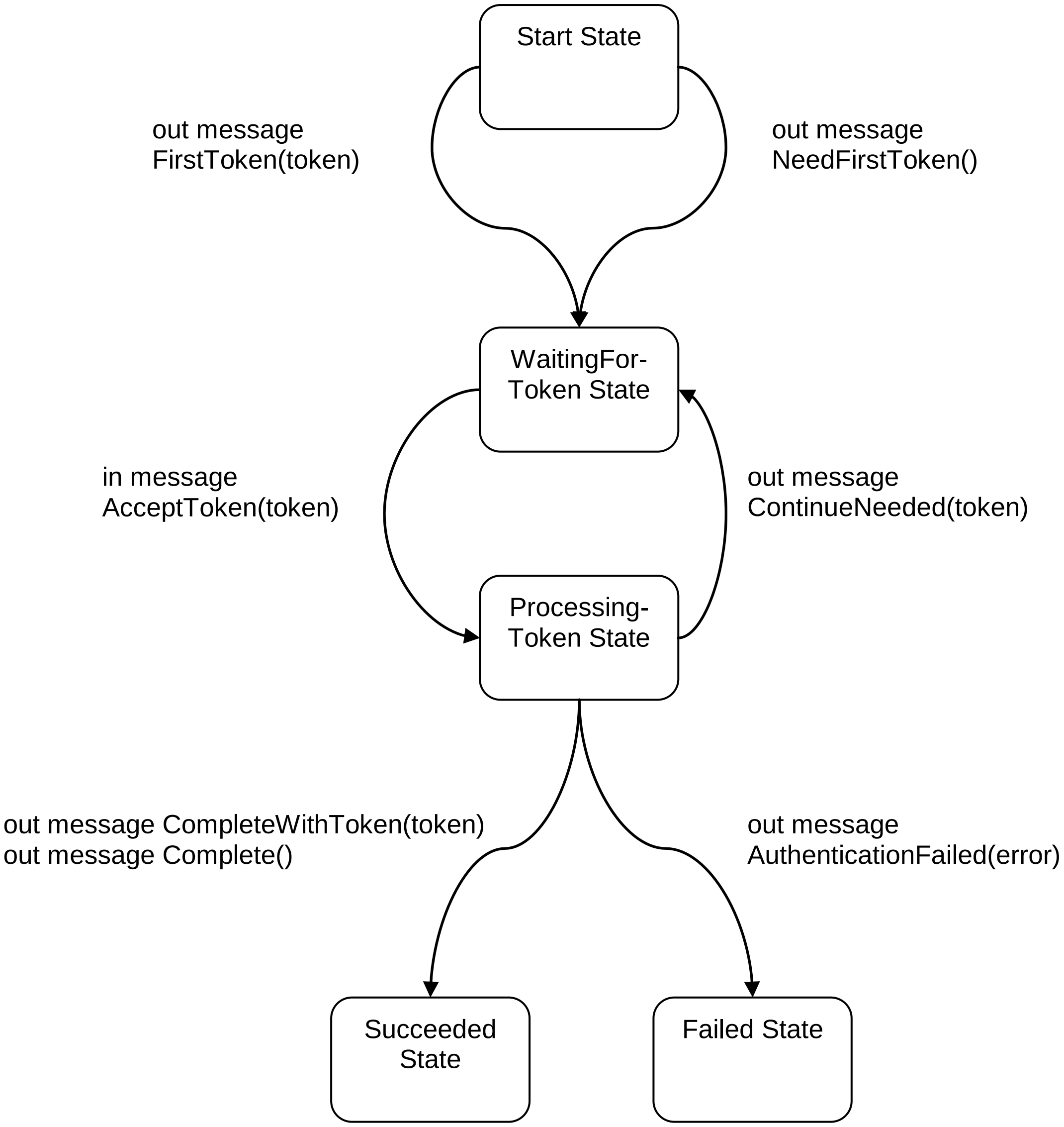
This contract models the "legacy" or raw NTLM interface, not the NTLMSSPI interface, which uses the `GssSupplicantContract`.



3.1.3. GssSupplicantContract

This contract allows applications to use any authentication protocol that supports the GSS model. GSS is described in RFC 2078, and is a platform-neutral and language-neutral model for authentication protocols. Authentication protocols exchange opaque tokens, which the applications exchange until both sides decide that authentication has succeeded, or one side decides that authentication has failed.

Many authentication protocols support GSS, include Kerberos, NTLM, and the SPNEGO meta-protocol.



3.2. The Credentials Manager Service

The CM Service is a singleton service process. It manages the in-memory Credentials Store and Protocol Mappings table. It also contains the client-side implementations of the authentication protocols.

3.3. The Credentials Manager Library

The Credentials Manager Library simplifies connecting to the CM Service and sending requests to it. Most of the requests defined on the `CredentialsManagerContract` are exposed as static methods. The library handles translating between exchange types and local heap types.

3.4. Supported Authentication Protocols

At present, only a single authentication protocol, NTLM, is supported. The Credentials Manager is designed to support other authentication protocols, as they are implemented.

3.4.1. Windows NT / LAN Manager (NTLM)

NTLM is an old authentication protocol, with some known cryptographic weaknesses, but it is an important protocol because it is so widely supported, and so easy to implement. Consequently, it is the first authentication protocol supported. The NTLM implementation supports two supplicant contracts: `NtlmSupplicantContract` and `GssSupplicantContract`.

The `NtlmSupplicantContract` allows network applications to deal directly with the NTLM challenges and responses.

The `GssSupplicantContract` allows network applications to exchange NTLMSSP messages. This provides the greatest degree of isolation between the network application and the authentication protocol. The network application is responsible only for exchanging opaque byte sequences, called tokens.

3.4.2. SPNEGO

SPNEGO is a standard for allowing network applications to negotiate among different authentication protocols. SPNEGO is not itself an authentication protocol. This protocol is not yet implemented.

3.4.3. Kerberos

Kerberos is the favored authentication protocol of Windows domains. It is not yet implemented.

3.5. Command-Line Tool

Singularity currently has no user session management or authentication. Until it does, users need some way to control network authentication. The CM package includes a command-line tool, cred.exe, which allows users to manage credentials and protocol mappings.

The tool implements the following commands.

3.5.1. Add Credentials

```
cred @add [-default=false] [-tag=<tag>] <username> <password>
```

This command submits user credentials to the Credentials Manager. This allows client applications to make use of credentials.

At this time, the only form of evidence that the Credentials Manager supports is clear-text passwords. It is anticipated that support for private keys and other forms of authentication will also be added.

This command also adds a default protocol mapping, which matches all protocol tuples, for these credentials. This is done as a convenience, since it is expected that most Singularity users, in the near future, will be using only a single set of credentials. If you do not wish to add a default protocol mapping, set the "default" parameter to false by adding `-default=false` to the command line.

The "tag" parameter allows the user to add multiple credentials that have the same username to the credentials store. This is useful if you need to authenticate with more than one machine, and have the same account name on those different machines, such as "Administrator" or "Test". The tag value is arbitrary. All requests that manipulate credentials take a tag value.

3.5.2. Delete Credentials

```
cred @del [tag-<tag>] <username>
```

This command deletes a credentials entry from the store. Applications can no longer use the credentials.

If a tag value was specified when the credentials were originally added, then the same tag value must be specified with this command.

3.5.3. Delete All Credentials

```
cred @delall
```

This command deletes all credentials in the store.

3.5.4. List Credentials

```
cred @list
```

This command displays a list of the credentials in the store. Evidence is not displayed.

3.5.5. Add Protocol Mapping

```
cred @addmap <app-proto> <service-addr> <auth-proto> <realm>
<credentials> [<tag>]
```

This command adds a protocol mapping to the protocol mapping table. Protocol mappings are policies that allow applications to choose which credentials to use when authenticating with a specific remote network service. Each protocol mapping consists of a protocol tuple, which describes when the mapping should be used, and the name of the credentials that should be used (and an optional credentials tag).

The protocol tuple consists of four fields:

- Application Protocol – This is the name of the application protocol, such as “smb”, “http”, “ftp”, etc. The Credentials Manager does not impose any requirements on this field, but these values should be chosen from a standardized list.
- Service Address – This is the network address of the remote network service. The nature of this field depends on the application protocol. The Credentials Manager does not interpret the contents of this field, so application designers are free to use whatever structure is appropriate. For SMB, this field is the server name portion of a UNC address. For example, for the UNC `\\server\share`, the Service Address field would be `server`.
- Authentication Protocol – This is the authentication protocol that the network application is using. The only protocol currently supported is NTLM, identified by the string `ntlm`.
- Realm – This is the authentication realm, which may or may not be known or discovered by the application protocol. The meaning of this value depends on the authentication protocol. For NTLM, this is a domain name, such as `WEBDEV`. For Kerberos, this would be the DNS name of the realm, such as `northamerica.fabrikam.com`. For some application protocols, this value may be learned during connection negotiation. For others, it may be unknown or unavailable, and in these cases, applications can use the wildcard value “*”.

Any of these fields may be set to the wildcard value “*”. Protocol tuples with fields set to wildcards will match any request for credentials where the non-wildcard fields match. This enables the most common scenarios, where a set of credentials is intended to be used for all machines in a particular domain, or for all connections to a particular server, etc.

Examples:

```
cred @addmap * * * * domain\user
```

This command will add a default mapping (all protocol tuple fields set to wildcard), whose result is the credentials “domain\user”. The credentials must have already been provided, using the @add command.

```
cred @addmap * 192.168.0.10 * * .\test
```

Adds a mapping for all connections to 192.168.0.10.

```
cred @addmap * * * northamerica northamerica\joe
```

Adds a mapping that is used whenever authenticating with a machine in the domain `northamerica`.

3.5.6. Delete Protocol Mapping

```
cred @delmap <app-proto> <service-addr> <auth-proto> <realm>
```

This command deletes a mapping that was previously added. The tuples must match exactly.

3.5.7. List Protocol Mappings

```
cred @listmap
```

This command shows all entries in the protocol mapping table.

3.5.8. Test Protocol Mappings

```
cred @testmap <app-proto> <service-addr> <auth-proto> <realm>
```

This command allows you to test the protocol mapping table. You specify a protocol tuple, and the command shows you the credentials that are selected for that tuple.

4. Performing Authentication Exchanges

When an application wishes to connect to a network service, it often needs to prove that it is acting on behalf of a specific user. There are many authentication protocols, and they differ in the number of messages exchanged, structure of the messages, etc.

Applications that use the CM Service for authentication go through these steps:

1. Decide which credentials to use. If the application is configured to use a specific set of credentials, then go to step 2. Otherwise, the application uses the CM Service to choose the appropriate credentials. The application provides these fields:
2. Create a supplicant, which is an instance of an authentication protocol. The application specifies the name of the authentication protocol, such as “NTLM” or “Kerberos”, and provides an endpoint used to communicate with the supplicant. The application creates the supplicant channel, and uses a contract that is specific to the kind of supplicant, e.g. `NtlmSupplicantContract` or `GssSupplicantContract`. The application should use either the `CredentialsManager.CreateSupplicant` or `Credentials-Manager.CreateSupplicantForProtocol` method, which are in CredentialsManager.Library.dll, to create and bind the supplicant channel.
3. Exchange messages with the network service and the supplicant channel. The nature of the messages and the exchanges is specific to the application and authentication protocols.
4. At some point, the application decides whether authentication has succeeded or failed, and deletes its channel to the supplicant.