Network Working Group
Internet-Draft
Intended status: Informational
Expires: October 13, 2018

P. Hallam-Baker
Comodo Group Inc.

April 11, 2018

Mathematical Mesh: Reference Implementation
draft-hallambaker-mesh-developer-07

Abstract

The Mathematical Mesh ?The Mesh? is an end-to-end secure infrastructure that facilitates the exchange of configuration and credential data between multiple user devices.

This document describes the Mesh reference code and how to install, run and make use of it in applications. It does not form a part of the Mesh specifications and is not normative.

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-mesh-developer.html [1] .

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 13, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Definitions 3
1.1. Requirements Language 3
1.2. Defined Terms 3
1.3. Related Specifications 3
1.4. Implementation Status 3
2. Getting the Reference Code and Build Tools 3
2.1. Obtaining the Development Environment 4
2.2. Obtaining the Build Tools 4
2.3. Obtaining the Mesh Source Libraries 5
3. Compiling the Reference Code 5
3.1. Creating a software signing key 5
3.2. Create (dummy) build action files 6
4. Running the Reference Code Examples 7
4.1. Starting the Server 7
4.2. The Profile Manager Wizard 7
4.3. The Profile Connection Wizard 8
5. Platform specific configuration data 8
5.1. Windows 8
5.1.1. Private Key Data 8
5.1.2. Registry settings 8
5.1.3. Profile data files 9
5.2. OSX and Linux 9
6. Using the Mesh C#/.Net Libraries in an Application 9
6.1. Portals, Sessions and Clients 10
6.1.1. MeshSession vs PersonalSession 10
6.2. Creating a Mesh Session 11
6.3. Creating a Mesh Session for Testing 12
6.4. Checking that a Portal Account name is acceptable 13
6.5. Creating a Personal Profile 13
6.6. Creating an Offline Escrow Entry 14
6.7. Deleting Profile Data 14
6.8. Recovering Profile Data 14
6.9. Connecting a New Device 15
6.10. Managing Applications 16
7. Using other languages 16
7.1. Lightweight API 16
8. Implementation Status 17
8.1. Reference Implementation 17
8.1.1. Coverage: 17

8.1.2. Licensing 18
8.1.3. Implementation Experience 18
8.1.4. Contact Info 18
9. Security Considerations 18
10. IANA Considerations 18
11. Acknowledgements 19
12. References 19
12.1. Normative References 19
12.2. Informative References 19
12.3. URIs 19
Author's Address 19

1. Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.

1.1. Requirements Language

This document is not normative and does not contain requirements language

1.2. Defined Terms

The terms of art used in this document are described in the Mesh Architecture Guide [draft-hallambaker-mesh-architecture] .

1.3. Related Specifications

The architecture of the Mathematical Mesh is described in the Mesh Architecture Guide [draft-hallambaker-mesh-architecture] . The Mesh documentation set and related specifications are described in this document.

1.4. Implementation Status

The implementation status of the reference code base is described in the companion document [draft-hallambaker-mesh-developer] .

2. Getting the Reference Code and Build Tools

The Mesh Reference library was developed using Visual Studio 2017 Community Edition [VS2017] using PHB?s Build Tools [PHB2017] extensions. The reference code itself is currently limited to C# libraries.

The code should in theory run under other operating systems but this has not been tested recently.

Development under different development environments is also possible but would require re-engineering to make use of the line mode versions of the build tools.

2.1. Obtaining the Development Environment

Visual Studio 2017 Community Edition is currently available at no cost for a wide range of non-commercial development including personal use and development of Open Source software. For full details, please consult the license published by Microsoft.

https://www.visualstudio.com/

Figure 1

2.2. Obtaining the Build Tools

Over half the code in the reference code library is generated using code generators. These are used to ensure that the specification, examples and reference code are always kept in synchronization.

The build tools are published under an MIT License and are available in two forms:

As stand-alone tools to be run from the command line.

As a VSIX package that integrates into the Visual Studio environment.

The source distribution is configured to use the tools integrated into the Visual Studio environment. If development on other platforms is desired, the simplest approach is likely to be to write a tool that reads the Visual Studio configuration files and generates the corresponding files for use with make.

The VSIX package is available from the Visual Studio extensions gallery:

PHB Code Generation Tools

Figure 2

The source code for the build tools is available from:

https://sourceforge.net/projects/phb-build-tools/

Figure 3

2.3. Obtaining the Mesh Source Libraries

The Mesh reference library source code is published under an MIT license and is available from:

https://sourceforge.net/projects/mathematicalmesh/

Figure 4

3. Compiling the Reference Code

To compile the code it is necessary to

Create a signing key

Create batch files for pre and post build tasks

3.1. Creating a software signing key

The purpose of signing assemblies is so that they can be authenticated during the load process. For this to be secure, it is of course essential that each developer has their own key.

To create a software developer signing key, the Visual Studio 'sn' tool is used. To run the tool, start the Visual Studio Developer Console in administrator mode. This requires the following steps:

Move to a directory you want to write to.

Set the machine to create user based keys

Create a new key and write it to a file.

Copy the file from the key to a container.

Delete the container.

Locate the private key file

Give permission to use the key.

This is of course one of the tasks we would like to automate with the Mesh tools in due course but that presents a bootstrap problem.

```
C:\Windows\System32>cd hallam

c:\Users\hallam>sn -m N

Microsoft (R) .NET Framework Strong Name Utility  Version 4.0.30319.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Key containers are user based

c:\Users\hallam>sn -k fred.snk

Microsoft (R) .NET Framework Strong Name Utility  Version 4.0.30319.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Key pair written to fred.snk

c:\Users\hallam>sn -i fred.snk SigningKeyDeveloper

Microsoft (R) .NET Framework Strong Name Utility  Version 4.0.30319.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Key pair installed into 'SigningKeyDeveloper'

c:\Users\hallam>del fred.snk

c:\Users\hallam>
```

Figure 5

3.2. Create (dummy) build action files

Visual Studio allows projects to specify batch files to be run before and after a project build. Since the actions to be taken are likely to change from developer to developer, these are specified in separate batch files. All that is necessary to build the code without warnings is to specify a set of dummy batch files with the following names and place them somewhere in the command line $PATH environment variable.

The files required are:

VSPreBuild.bat

VSPostBuild.bat

VSPostBuildWindows.bat

VSPostBuildOSX.bat

```
VSPostBuildLinux.bat

The following code will prevent error messages being thrown:

@echo off
SETLOCAL
exit /b 0
```

Figure 6

4. Running the Reference Code Examples

The reference code examples are designed to illustrate how the Mesh might be used in an application rather than be standalone tools in their own right. The Mesh is designed to make it each for developers to add security to their own applications rather than providing the applications themselves.

4.1. Starting the Server

On the Windows platform, the server runs in the context of the platform Web server and must be granted permission to bind to the range of server addresses used using the netsh command.

From a command prompt with administrator privileges, run the following command:

```
netsh http add urlacl http://<domain>/.well-known/mmm/
    \user=<machine>\<user>
```

Figure 7

Where is the DNS domain name under which the service is run, is the Windows domain name of the machine and the account name.

To start the service from the command line type:

```
servermesh <domain>
```

Figure 8

The server does not require administration privileges.

4.2. The Profile Manager Wizard

The profile manager wizard demonstrates functions that are performed on an administration device. These include creating a completely new

profile and initial configuration of applications, connecting a device to the profile and recovery of the profile from escrow data.

To run the client from the command line, place the executable image in a location that it will be found in the PATH variable and type:

meshclient

Figure 9

4.3. The Profile Connection Wizard

The Profile connection wizard demonstrates the much more restricted functionality that would be required in a Mesh connected application and/or a profile manager for a non-administration device.

To run the client from the command line, place the executable image in a location that it will be found in the PATH variable and type:

meshconnect

Figure 10

5. Platform specific configuration data

5.1. Windows

5.1.1. Private Key Data

All private key data is stored using the Windows public key store. At minimum, this ensures that private keys are obfuscated and encrypted under the account password to protect the data against casual extraction attacks. On a machine with cryptographic hardware support such as a TPM or HSM, extraction of the private key may be infeasible without physical access to the machine and possibly require sophisticated diagnostic equipment.

5.1.2. Registry settings

Separate settings are used for production and test code. Test Code should use the Registry Hive:

HKEY_CURRENT_USER\SOFTWARE\CryptoMesh

Production code should use the hive

HKEY_CURRENT_USER\SOFTWARE\MathematicalMesh

In either case the sub structure is:

Accounts Contains the set of Mesh Portal Accounts for the user. The default value is the account name of the default account. The Name of the each key is a portal account name and the value a REG_SZ entry containing the UDF of the profile master key.

PersonalProfiles Contains the set of Mesh Profiles for the user. The default value is the UDF of the default profile master key. The Name of each key is the UDF of the master key and the value a REG_SZ entry containing the file location of the cached copy of the personal profile.

ThisDevice Contains the set of Device profiles in the same format as the PersonalProfiles.

5.1.3. Profile data files

The profile data itself is stored in data files at the location specified in the registry. The files are standard XML files in UTF8 encoding.

5.2. OSX and Linux

[[Not yet implemented, subject to change.]

All configuration information is stored in the user directory ~/.mmm

Keys are stored in SSH key file format [RFC4716] using the customary name and extension conventions for that application.

6. Using the Mesh C#/.Net Libraries in an Application

The application ExampleGenerator shows the use of the Mesh in an application using the convenience API. It is the application program used to generate the examples in the reference document.

ExampleGenerator implements a client that connects to a remote Web Service, creates new personal profile with an escrow entry with offline recovery codes, attaches applications and other devices, updates an application profile, deletes all the profile data from the local machine and then restores them using the recovery codes and escrow entry.

6.1. Portals, Sessions and Clients

The libraries are designed to support testing and development use. For this reason, the client side of the libraries is divided into the following main classes:

MeshClient Provides a logical connection to a remote or simulated Mesh service.

MeshPortal Provides the interface to a Mesh service which may be an actual remote service accessed via a network connection, or local code running in the same process as the client to simulate a Mesh service for testing purposes.

MeshMachine Provides an interface to Mesh data stored on the local machine.

MeshSession / PersonalSession Provide the high level application interface to the Mesh combining access through the MeshClient and MeshMachine.

The relationship between these parts is shown in . The application programmer will typically need only the MeshSession class.

The principal classes in the Mesh Client side API.

This division makes it possible to test Mesh clients and server implementations in a single process with a single debugger which is usually more convenient than spinning up a separate development session for the client and service.

6.1.1. MeshSession vs PersonalSession

Most Mesh operations are performed within the context of a specific PersonalProfile registered on the current machine. This context is provided by an instance of the PersonalSession class.

An instance of the MeshSession class is used for operations that are not bound to a specific PersonalProfile registered on the machine. These operations are:

- Binding a new PersonalProfile to the machine.
- Offline key recovery.
- Requesting and completing a device connection request from the new device.

- Acquiring a PersonalSession instance.

6.2. Creating a Mesh Session

The primary interface for the application programmer is the MeshSession class. To create a mesh session class, the following steps are required:

1. Initialize the Mesh code for the intended platform

2. Request a new MeshSession instance.

Although C# code is nominally 'write once, run anywhere', this approach does not ensure use of platform specific features such as the Windows registry or protected storage for cryptographic keys. Calling MeshWindows.Initialize() causes the platform specific code for the Windows to be initialized in production mode. Alternatively, calls to MeshLinux.Initialize() or MeshOSX.Initialize() causes the platform specific code for those platforms to be initialized.

The code to initialize a production instance of the code is shown in :

```
static MeshSession MeshSession = null;

static void ApplicationInit () {
    MeshWindows.Initialize();
    MeshSession = new MeshSession();
    }
```

Figure 11

If the user has already created a PersonalProfile and connected it to the machine, it will automatically be read from local storage. The instance will automatically create MeshClient instances as required to establish a web service using the default transport (HTTP) to the service as necessary (see).

Connecting to a remote service from a Windows platform.

The server implementation is managed in the same fashion. Internally, the MeshService and MeshClient classes are both descended from the same parent.

6.3. Creating a Mesh Session for Testing

Since the purpose of the ExampleGenerator is to create examples for the documentation, it is not necessary for the JSON Remote Procedure Calls to actually be ?Remote?. Instead the ?Local? Procedure Call mode is used in which the client and server both run in the same process with the client API invoking the server dispatch methods through an interface that performs JSON serialization and deserialization but does not invoke the network transport.

Connecting to a direct service for testing.

A direct connection to the service provider may be established by either specifying the portal to use in the initialization of MeshSession or by setting the default portal property of the MeshPortal class as is done here .

```
        static void DebugApplicationInit () {

            MeshPortal.Default = new MeshPortalDirect("example.com",
                "MeshLog.jlog", "PortalLog.jlog");

            MeshWindows.Initialize(true);
            MeshSession = new MeshSession();
            MeshSession.EraseTest();
            }
```

Figure 12

This time, we initialize a specific version of the platform dependent code and specify that it is to be initialized as test code rather than production. This will cause all persistent data stored on the machine (keys, profiles) to be stored in locations marked as test locations. The EraseTest() method causes all data stored in test locations to be erased from the machine, thus ensuring that the test begins from a known state with no results from previous runs.

When writing test code, it is frequently useful to create multiple independent MeshSessions to simulate multiple machines. To prevent data written to one machine interfering with another, a new simulated machine is created for each session using the MeshMachineCached class

```
            MeshSession = new MeshSession(new MeshMachineCached());
```

Figure 13

6.4. Checking that a Portal Account name is acceptable

The user experience is improved if the application indicates whether their choice of portal account name is acceptable or not while they are entering it. The Validate method allows the user's choice of account name to be validated .

```
PersonalProfile PersonalProfile;
PersonalSession PersonalSession;
OfflineEscrowEntry OfflineEscrowEntry;

void DebugCreateProfile () {
    var Response = MeshSession.Validate("alice@example.com");
    if (!Response.Valid) {
        throw new Exception();
        }
    ...
```

Figure 14

The portal address is given in the usual username@domain format, for example alice@example.com.

6.5. Creating a Personal Profile

Creating a PersonalProfile has two steps:

1. Create a DeviceProfile (if necessary)

2. Create the PersonalProfile

3. Create an account bound to the profile at the portal.

These steps are shown in .

```
var Device = MeshSession.CreateDevice();
PersonalProfile = new PersonalProfile(
    Device.DeviceProfile);
PersonalSession = MeshSession.CreateAccount(
    "alice@example.com", PersonalProfile);
```

Figure 15

The application could have overridden the default values of DeviceID and DeviceDescription when creating the device.

6.6. Creating an Offline Escrow Entry

Having created a potentially valuable profile, we probably want to back it up. To do this, we create an instance of the OfflineEscrowEntry class with the desired quorum and number of shares (2 out of 4) .

```
OfflineEscrowEntry = new OfflineEscrowEntry(
    PersonalProfile, 2, 4);
PersonalSession.Escrow(OfflineEscrowEntry);
```

Figure 16

6.7. Deleting Profile Data

We can test our escrow parameters by deleting the profile from the current machine using the Delete method .

```
PersonalSession.Delete();
```

Figure 17

6.8. Recovering Profile Data

Profile recovery has two steps:

1. Reconstruct the shared secret from the recovery shares.

2. Recover the profile.

In this case our recovery shares are the first and the third key shares we just generated. The Recover method recovers the profile and rebinds it to the existing portal .

```
var RecoveryShares = new KeyShare[] {
    OfflineEscrowEntry.KeyShares[0],
    OfflineEscrowEntry.KeyShares[2] };

var Secret = new Secret(RecoveryShares);
PersonalSession = MeshSession.Recover(
    Secret, "alice@example.com");
}
```

Figure 18

6.9. Connecting a New Device

Device connection involves two devices, the device to be connected and the device used to approve the request.

The new device:

1. Create a device profile for the new device.

2. Request connection to the new device

3. Wait for the result.

These calls are shown .

```
        void RequestConnect (string Address) {
            var DeviceRegistration = MeshSession.CreateDevice();
            var Connect = MeshSession.Connect(DeviceRegistration,
                    Address, out var Authenticator);
            PersonalSession = Connect.Await();
            }
```

Figure 19

In a real example, we would want to show the connection authentication code to the user so that they can verify that they are responding to the right request on the approval device.

On the approval device, the application

1. Requests a list of pending requests using ConnectPending.

2. Accepts or Rejects devices using ConnectClose.

```
        void AcceptPending () {

            var Pending = PersonalSession.ConnectPending();
            foreach (var Request in Pending.Pending) {
                var Result = PersonalSession.ConnectClose(Request,
                    ConnectionStatus.Accepted);
                }
            }
```

Figure 20

6.10. Managing Applications

Application profiles are created in the same manner as personal profiles .

```
var PasswordProfile = new PasswordProfile(true);
var RegistrationApplication =
        RegistrationPersonal.Add(PasswordProfile, false);
```

Figure 21

Changes to the Application Profile are written to the RegistrationApplication instance and then committed using the Update() method.

7. Using other languages

If you are building Mesh applications in another language, the least effort approach may be to rewrite the PROTOGEN build tool to target your language.

Protogen does support generation of C header files that may be used to drive a parser. If however you are adding Mesh support for an application that already uses JSON based protocols, you might want to edit the generator scripting files to generate code for your existing libraries.

7.1. Lightweight API

A lightweight API providing the minimal features required to Mesh enable an application is required. Such an API should exclude most account management features:

- Creating new Personal Profiles and portal accounts.
- Key escrow, recovery
- List, accept pending device connection requests

This leaves the following features:

- Create Device Profile
- Request device connection
- Get Personal Profile
- Get, Update, Application Profile

In addition to providing less functionality, an implementation of the lightweight binding is likely to be written in a 'flattened' style rather than the abstracted, object oriented approach of the reference code.

8. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6892] . The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6892] , "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

8.1. Reference Implementation

Organization: Comodo Group Inc.

Implementer: Phillip Hallam-Baker

Maturity: Experimental Prototype

This implementation was used to produce the reference section and all the examples in this document. Since the conversion of specification to code is automatic, there is a high degree of assurance that the reference implementation is consistent with this document.

8.1.1. Coverage:

The draft-xx branch describes the code used to create version xx of this document.

The main current limitations are that the code only supports RSA key pairs and for ease of development the server does not persist keys across sessions. Nor does the implementation currently support the

HTTP payload authentication and encryption layer or make use of TLS. These could be easily fixed.

The client and server are implemented as libraries that may be called from a multi-protocol server. A standalone server will be provided in a future release.

Only the JSON encoding is currently implemented. The JSON-B, JSON-C, ASN.1 and TLS Schema implementations are all supported by the code generation tool but not currently implemented as the build tool bindings for those encodings have not yet been finalized or documented.

The key restrictions for TLS key exchange have not yet been implemented.

The code has only been tested on Windows 10 but passed compatibility testing for both Mono and dotNetCore 2.0 run times which should in theory permit use on Linux and OSX platforms.

8.1.2. Licensing

The code is released under an MIT License

Source code is available from GitHub at https://github.com/hallambaker/Mathematical-Mesh

8.1.3. Implementation Experience

The implementation and specification documentation were developed in Visual Studio using the PHB Build Tools suite.

8.1.4. Contact Info

Contact Phillip Hallam-Baker phill@hallambaker.com

9. Security Considerations

Security Considerations are addressed in the companion document [draft-hallambaker-mesh-architecture]

10. IANA Considerations

This document specifies no actions for IANA

11. Acknowledgements

Comodo Group: Egemen Tas, Melhi Abdulhayo?lu, Rob Stradling, Robin Alden.

12. References

12.1. Normative References

[RFC4716] Galbraith, J. and R. Thayer, "The Secure Shell (SSH) Public Key File Format", RFC 4716, DOI 10.17487/RFC4716, November 2006.

12.2. Informative References

[draft-hallambaker-mesh-architecture]
Hallam-Baker, P., "Mathematical Mesh: Architecture", draft-hallambaker-mesh-architecture-04 (work in progress), September 2017.

[draft-hallambaker-mesh-developer]
Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", draft-hallambaker-mesh-developer-06 (work in progress), April 2018.

[PHB2017] "[Reference Not Found!]".

[RFC6892] Wilde, E., "The 'describes' Link Relation Type", RFC 6892, DOI 10.17487/RFC6892, March 2013.

[VS2017] "[Reference Not Found!]".

12.3. URIs

[1] http://mathmesh.com/Documents/draft-hallambaker-mesh-developer.html

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com