Mathematical Mesh 3.0 Part VI: Mesh Presentation Layer Security

Mesh Presentation Layer

<series>draft-hallambaker-mesh-presentation

<status>informational

<stream>independent

<ipr>trust200902

<author>Phillip Hallam-Baker

<surname>Hallam-Baker

<initials>P. M.

<firstname>Phillip

<email>phill@hallambaker.com

<organization>ThresholdSecrets.com

<keyword>Threshold Cryptography

<keyword>Elliptic Curve

<keyword>Threshold Encryption

<keyword>Threshold Key Generation

<keyword>Ceremony

A presentation layer suitable for use in conjunction with HTTP and UDP transports is described.

Discussion of this draft should take place on the MathMesh mailing list (mathmesh@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/mathmesh/>.

# Introduction

The Mesh Presentation Layer (MPL) is a lightweight key exchange, authentication and encryption protocol that provides a transactional presentation when layered over a HTTP transport and transactional and streaming presentations when layered over a UDP transport.

The UDP binding of MPL provides much of the functionality normally provided through use of TCP, TLS, HTTP and Web Sockets with considerably less complexity. MPL does not attempt to reproduce all the functionality of these protocols, only the functionality required by Web Services and streaming applications is provided.

<figuresvg="../Images/PresentationTcpTlsHttp2Mpl.svg"/>Mesh Presentation Layer provides a combination of necessary functionality from TCP, TLS, HTTP and Web Sockets.

As recognized in the design of QUIC, combining these protocols allows greater efficiency and security. But QUIC is the product of one set of design choices optimized to meet an important but specific set of needs. MPL makes a different set of choices to meet a different set of needs. A natural consequence of moving TCP functionality from a platform capability implemented in the OS kernel to an application level resource it that applications are now free to chose the transport and security capabilities that best fit their needs rather than being limited to a single size fits all.

The reduction in complexity afforded by combining four protocols into one allows entirely new communications patterns to be supported that are poorly supported by traditional approaches. HTTP is built on the Remote Procedure Call model of transactions that consist of a single request followed by a response. Using this communication pattern, a device attempting to synchronize a local store with a store at a remote service is required to periodically poll the service for updates:

<figuresvg="../Images/PresentationHttpPolling.svg"/>Limiting communications to a request/response communication requires inefficient polling for updates.

MPL allows a client device with this particular requirement to subscribe to a continuous stream of updates. There is no need for the device to poll the remote store for updates as it will be notified every time an update occurs.

<figuresvg="../Images/PresentationMplSubscription.svg"/>MPL supports asynchronous streaming of updates as they occur.

While it is possible to achieve the same functionality in TCP, this requires that the device and service maintain a TCP connection for the duration of the connection. This consumes communication resources and operating system resources.

As is shown in later sections, the 'blank slate' design approach of MPL allows completely new approaches to service discovery, network agility, presence, NAT tunneling and many other requirements that are critical in modern network use.

## Background

Web Services have traditionally made use of HTTP bindings over TCP or TLS transports. While this approach is functional, it is less than satisfactory. The HTTP and TLS stacks are designed to meet the needs of Web browsing but fail to provide significant capabilities required by a service. Modern implementations of these stacks are large and cumbersome but little of that complexity addresses needs relevant to Web Services.

A basic requirement for a Web Service is the need to authenticate requests and the party making them. In particular, a Web Service requires the ability to authenticate the source of a request and verify that authentication of the request data is complete before acting on it. As its original name, 'Secure Socket Layer' implies, TLS is designed to present an abstract socket layer that hides this information from the application layer. In many large enterprise deployments, TLS processing and Web Service transactions are performed on separate machines and information from the TLS session is only visible to the transaction processor if special provision is made.

HTTP provides a wide range of protocol capabilities but almost none of this complexity is relevant to Web Services needs.

* Providing access to multiple Web Services through a single TCP/IP port.
* Delineating the beginning and end of request and response data.

Recognizing these needs, the DARE envelope format was originally conceived as a JSON equivalent of the XML Signature and Encryption protocols applied to provide message layer security in the WS-Security stack. Consideration of the resource requirements needed to verify each transaction request and sign each response led to the need for a key exchange mechanism and that this should be separate from the Mesh Service Protocol. This in turn led to the realization that in most circumstances, an AEAD scheme such as AES-GCM or AES-CFB would provide encryption with the same or less overhead than use of a MAC.

Having taken these considerations into account, a new lightweight envelope format was designed to allow authentication and encryption of Web Services messages as the payload of HTTP requests and responses. The success of this approach led to the realization that the same format could be modified to enable direct binding to UDP, allowing provision of HTTP support to be made optional.

## Protocol Elements

A TCP connection provides a single pair of bidirectional communication streams between two devices connected to the network. While the TCP connection is symmetric from the application layer point of view, the HTTP presentation layer is not. HTTP limits communications to a request/response communication pattern in which the initiator device makes all the requests and the responder device makes all the responses.

MPL affords greater communication flexibility requiring distinctions to be made between terms which may be used interchangeably in the HTTP over TCP/IP case.

Connection

An MPL connection is a bidirectional relationship between a pair of endpoints established through a cryptographic key exchange establishing a shared secret called the Connection Origin Key.

Stream

An MPL stream is an individual communication channel established within a connection. An MPL connection comprises one or more streams.

Port

An MPL port is a network interface through which data is exchanged. Currently only UDP and HTTP ports are supported. Each end of an MPL Connection accepts packets through one or more ports.

The distinction between connections and streams is analogous to that made between processes and threads. While any task that is performed by a separate thread may be performed by a separate process, use of threads consumes fewer platform resources and greater convenience to the programmer.

In the TCP/IP model, each stream is bound to a single pair of IP port. In the MPL model, any stream MAY make use of any available destination port to send traffic. This allows Alice to move from her home to her car to her office without dropping the connection to her conference call.

<figuresvg="../Images/PresentationConnectionsStreamsPorts.svg"/>

New streams may be created at any time by either party in a connection for many different reasons:

* To establish a client/service relationship to a different Web Service.
* To establish a new asynchronous streaming relationship
* To signal changes to the cryptographic context (e.g., forward secrecy rachet operation).
* Exhaustion of the frame counter identifier space.
* To make use of traffic analysis avoidance techniques.

A device connected to three separate Web Services provided by the same host establishes a separate stream for each service. This enables the device to present different application-level credentials top the different services. If permitted by the service, separate tasks on the same device MAY establish separate streams to the same service. This allows separate threads to wait on transactions taking a long time to complete without contention.

Frame is the minimum payload unit

In a video game, need to transmit player movements to others with minimal latency.

In a file archive application, transfer much larger chunks of data (64KB)

<figuresvg="../Images/PresentationStreamsFramesPackets.svg"/>

## Application Interface

The traditional TCP/IP programming API exposes only an undifferentiated stream of bytes to the application program context. Information such as packet latency, packet loss rate, error rate, etc. are only available through proprietary platform specific APIs, if at all.

MPL is designed to provide application programs with relevant connection information including:

* Time since the most recent forward secrecy exchange
* Average round trip latency (per port)
* Packet loss (per port)

Providing this information allows applications to adapt to changing network conditions.

* Forcing a forward secrecy exchange when a participant leaves a shared chat context
* Making use of an outbound port with a different physical network connection.
* Establishing communication to a different host.

Transactional interface, bidirectional, every request is followed by a response.

Provide feedback to the application, how long it takes to transmit a frame.

Allow pre-emption of a frame transfer. No point continuing to send an 8K frame if the available bandwidth is only found capable of supporting HD.

## Service Negotiation

<figuresvg="../Images/PresentationServiceAdvertisement.svg"/>

## Network Agility

<figuresvg="../Images/PresentationNetworkAgility.svg"/>

## Presence

<figuresvg="../Images/PresentationAlicePresenceBob.svg"/>

[Brokered port]

## NAT Transit

<figuresvg="../Images/PresentationNatChange.svg"/>

<figuresvg="../Images/PresentationBrokeredNat.svg"/>

## Broadcast

[broadcast ports]

<figuresvg="../Images/PresentationSatellite.svg"/>

## Tunneling

<figuresvg="../Images/PresentationOnion.svg"/>

# 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.

## Requirements Language

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 <norm="RFC2119"/>.

## Defined Terms

## Related Specifications

## Implementation Status

The implementation status of the reference code base is described in the companion document <info="draft-hallambaker-mesh-developer"/>.

# Architecture

Data Encoding

[This part of the specification describes the encodings to be used in the next release. The data encodings in the current reference code used to generate the examples do not match. These should be aligned with QUIC where appropriate. ]

Encoding at the packet level presents different requirements to those at higher levels. These needs are better met by the compactness afforded by a place-value approach than the JSON/JSON-B Tag-Value encoding used in the upper layers of the Mesh specifications.

As at the application layer, the use of nested length encodings is avoided on account of the risk that bounds overflow vulnerabilities are introduced into applications.

Integer

Integer values are presented using the same variable length encoding described in section 16 <norm="draft-ietf-quic-transport">

The first two most significant bits of the first byte are reserved to encode the base 2 logarithm of the integer encoding length in bytes. The integer value is encoded on the remaining bits, in network byte order.

|  |  |  |  |
| --- | --- | --- | --- |
| 2 Bit | Length | Usable Bits | Range |
| 00 | 1 | 6 | 0-63 |
| 01 | 2 | 14 | 0-16383 |
| 10 | 4 | 30 | 0-1073741823 |
| 11 | 8 | 62 | 0-4611686018427387903 |

Values do not need to be encoded on the minimum number of bytes necessary.

String

String values are Unicode strings encoded as a sequence of UTF-8 bytes preceded by an integer length specifier.

Binary data

Binary data consists of a sequence of opaque bytes encoded as the sequence of byte values preceded by an integer length specifier.

Extension List

Extension lists are used

Extension Tag

Stream Identifier

Every MPL packet begins with a **stream identifier**. Stream identifiers consist of an opaque sequence of zero to 16 bytes.

Stream identifiers are used to allow their issuer to distinguish packets arriving from different streams. Initiators and responders both issue stream identifiers for their own use during Connection Establishment. Additional stream identifiers MAY be issued as a means of defeating some forms of traffic analysis attack.

The semantics and encoding of stream identifiers including determining the length of the stream identifier is sole concern of the issuer which MUST ensure that it can correctly interpret the stream identifiers they issue for streams received on a specified port without ambiguity.

Stream identifiers are issued as binary data sequences specified in Connection Establishment packets and as Packet Extensions in Data Packets.

Issuers MUST accept every stream identifier issued as a valid identifier for that stream for the lifetime of the connection in which the stream was created.

## Connections

An MPL connection is a bidirectional relationship between two devices secured by a shared secret established through a key agreement between the devices.

Multiple streams - stream = thread.

Avoid the need to multiplex within a stream by simply creating streams as required.

Multiple ports - these may shift during a connection.

## Ports

A network communications resource. Currently only HTTP and UDP ports are considered. The data formats have been chosen so as to enable future extension to support additional port types, in particular non-IP LAN technologies (Bluetooth, Zigbee, etc.) and peripheral attachment (USB, HDMI, etc.)

High bandwidth, high latency combined with low bandwidth low latency.

## Stream

Streams have separate identifiers for both directions. Both are allocated by the receiver and unique for the party allocating them.

### Transactional stream

Each request is followed by a response.

<figuresvg="../Images/PresentationStreamTransactional.svg"/>

### Asynchronous stream

Stream of frames which do not normally receive any application level interaction.

<figuresvg="../Images/PresentationStreamAsynchronous.svg"/>

### Broadcast

Consists of a unidirectional port paired with a bidirectional port used for transmission control, request resending of individual packets, etc.

<figuresvg="../Images/PresentationStreamBroadcast.svg"/>

## Credentials

MPL follows the Mesh approach of using separate credentials for devices and principals.

For example, the service offered by provider (@provider) is delivered by a fault tolerant collection of three separate hosts. Alice (@alice) can access those services through any one of her four personal devices.

<figuresvg="../Images/PresentationLayeredCredential.svg"/>

MPL makes use of the keys specified in device credentials to perform key agreement and enables user and service credential(s) to be presented for authentication and authorization of the principals.

### Device

An MPL device credential is a credential that specifies a public key of one of the supported key agreement algorithms. Two forms of device credential are currently defined:

* Raw Key (e.g. X.448 key)
* Mesh Device Connection Assertion.

It might well be the case that it is more appropriate to pass the Device Connection Assertion as part of the Principal Credential.

### Principal

Principal credentials are used to authenticate and authorize a device to act on behalf of the specified principal.

Presentation of Principal Credentials MAY be performed at the application layer (i.e. by means of credentials carried in an MPL payload) or as an MPL Packet Extension when a **stream service binding** is requested.

In either case, validation of Principal Credentials is an application concern and thus outside the scope of MPL.

Different streams MAY involve the use of different principal credentials. Alice might be authenticated through her Mesh profile in one stream, a PKIX client cert bound to a smartcard in another and a SAML authorization token in a third.

## Data Frame Format

An MPL Frame is by definition the smallest unit of payload data passed from the application layer to the presentation layer or vice versa. A frame MAY be as large as a complete frame of 8K video or as small as a single key press or movement of a game controller.

The HTTP binding places no upper limit on the size of an MPL frame but in the Datagram binding, frames are limited to a maximum number of packets. Larger

<figuresvg="../Images/PresentationFrame.svg"/>

### Stream Identifier

A sequence of bytes.

Issuer is responsible for interpreting including determining length.

May determine length by issuing StreamIDs of fixed size or make use of the self describing length scheme of the issuer's choice.

In certain situations where traffic analysis is of particular concern, an issuer may issue 'one time' stream ids intended for single use.

### Initialization Vector

Frame sequence index

When a frame is comprised of multiple packets, a dummy IV consisting of just the frame sequence index is used.

### Ciphertext

Ciphertext is the plaintext payload encrypted under a key derived from the active primary key and the initialization vector. Keys are only used to encrypt a single frame or continuation frame.

Plaintext of a data frame consists of a list of packet extension entries followed by a payload followed by zero padding.

### Authentication Tag

Associated data is entire frame from the first byte to the start of the Initialization vector.

## Packet Format and Packetized Frames

UDP provides unreliable transport of datagrams no larger than the maximum IP payload size of 64KB. In practice the largest datagram payload is typically limited to no more than 1260 bytes because the Ethernet specification is stuck in the Middle Ages. It is thus necessary to split large frames into multiple packets and provide sufficient control information that the receiver can reassemble the packets in the right order and request any lost packets be re-sent.

Similar constraints are imposed by a wide range of wireless and wired physical and network transports.

When using Packetized frames, each packet in the frame specifies the stream identifier and a packet identifier. Only the last packet in the frame presents an authentication tag.

<figuresvg="../Images/PresentationPacketizedFrame.svg"/>

The initialization

### Packet Identifier

Frame sequence index

Packet count

Packet sequence index

Packet frame index [if required]

Continuation frame flag.

Acknowledgement bits

### Flow control

TCP flow control is designed to meet the needs of the day. Modern networks have very different needs.

Congestion avoidance is still important and implementations MUST take steps to avoid overburdening network resources.

May require additional packets that only contain flow control info. This MAY be used to communicate One time Stream IDs.

Congestion control limits MAY be specified for the connection as a whole and for individual ports. In situations where multiple Internet connections are available, applications MUST enforce flow control limits on each port.

<figuresvg="../Images/PresentationFlowControl.svg"/>

Senders continuously monitor response packets to determine the

### Partial Frames

When a Frame is too large for the number of packets, partial frames are used. A flag on the first frame specifies that it is a partial frame. The partial frame has its own unique initialization vector (and hence encryption key).

The principal consequence of the use of partial frames is that instead of there being one authentication tag for the whole frame, each partial frame has its own authentication tag.

## Connection Frames

Connection frames are used during the key exchange used to establish a connection. Unlike data frames in which have a single data section, all of which is encrypted, connection frames may contain three separate types of data section:

Plaintext data

Unencrypted data section. This is used to pass the information needed to perform the 'mezzanine' key exchange to the responder's credential and to present and respond to challenges.

Mezzanine data

Data encrypted under 'mezzanine' key exchange authenticated by the responder's credential alone. This is used to pass information related to the initiator credential.

Ciphertext data

Data encrypted under the mutual key exchange authenticated to the credentials of the initiator and the responder.

The ClientCompleteDeferred frame contains all three data sections. The ciphertext data section is encapsulated inside the Mezzanine data section and is thus encrypted twice. Only the Mezzanine section is padded.

<figuresvg="../Images/PresentationConnectionFrame.svg"/>

A connection frame MAY contain a payload. This is always carried in the last data section.

Connection frames that only contain plaintext data MUST NOT contain confidential data. This requirement may be enforced in a future revision of the specification by limiting use to a 'service discovery' protocol.

# Connection Establishment and Maintenance

## Key Exchange modes

### First Contact

<include=..\Examples\PresentationFirstContact.md>

### Zero Round Trip

<include=..\Examples\PresentationZeroRoundTrip.md>

### Deferred First Contact

<include=..\Examples\PresentationFirstContactDeferred.md>

### One Round Trip

<include=..\Examples\PresentationZeroRoundTripDeferred.md>

## Ticketed Connection

For supporting presence and discovery services.

Ticket specifies a shared secret, session ID, network end point

Example discovery, Alice's device connects to the provider's discovery service, is authenticated and receives a ticket to connect to a specific host. This allows a connection to be established to that host without the need to perform a connection key exchange.

Example presence, Alice wants to talk to Bob, contacts his discovery service, gets a ticket back allowing Bob to establish a direct, encrypted connection.

In either case, a client SHOULD perform a rekey operation as indicated by requirements for perfect forward secrecy, security policy, yada yada.

## Connection Forward Secrecy Rekeying

Rekeying is only necessary to provide forward secrecy. The data encrypted under a single symmetric key is a small fraction of the maximum. But a connection that persists for several years still represents a liability.

[Should we reintroduce the mezzanine packet so we can use the remainder of a packet without rekeying?]

[Packet extension contains the rekey data]

# Stream Establishment and Maintenance

## Stream Creation and Binding

When a stream is created, it is typically bound to a service.

A host might offer Mesh, Callsign and Discovery services. After connecting to the host, a device requests a stream to each of the services in turn. Each stream MAY be separately authorized.

## Stream Close

## One Time Stream Identifier

Can use a block cipher to construct one time IDs that require no state to interpret. The issuer generates a symmetric key that is fixed for the duration of the connection. To create a new one time StreamID, the issuer encrypts a block containing the stream ID to which the encrypted Id is to be mapped and a salt (e.g. a unique sequence number)

To interpret a one time ID, the service simply decrypts the encrypted form to recover the original stream ID, the salt is ignored.

Note that when using this approach it is absolutely critical that the cipher used is a block cipher and not a stream cipher or a block cipher used in a streaming mode (e.g. AES-GCM).

<figuresvg="../Images/PresentationOneTimeStreamId.svg"/>

<figuresvg="../Images/PresentationOneTimeStreamIdRecharge.svg"/>

## Stream Rekey

Rekeying the stream requires us to change the stream ID. No other characteristics of the stream are changed.

Stream can rekey by specifying its own rekey data or say 'use the same keydata as this stream Id'.

Thus a connection with four separate streams only needs to perform one rekey to refresh the crypto for all four.

# Schema

## Connection Frames

## Packet Extensions

# Security Considerations

# IANA Considerations

This document requires no IANA actions.

# Acknowledgements